X11 Window System
Core Distribution

Volume I

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

490243 Rev. A

September 1988

PREFACE

This manual is based on the Core Distribution Documentation from MIT. It is divided into two volumes as follows:

Volume I contains documentation on:

- Protocol
- Xlib
- Intrinsics
- Widgets
- Conventions

Volume II contains documentation on:

- Andrew Toolkit
- Xrlib Toolbox

Although ISI provides documentation and software for Andrew Toolkit and Xrlib Toolbox, ISI does not offer technical support for these packages.

X Window System Protocol

Release 2

X Version 11

Robert W. Scheifler

Massachusetts Institute of Technology
Laboratory for Computer Science

Table of Contents

Acknowledgments

The primary contributers to the X11 protocol are:

Dave Carver (Digital HPW)
Branko Gerovac (Digital HPW)
Jim Gettys (MIT/Project Athena, Digital)
Phil Karlton (Digital WSL)
Scott McGregor (Digital SSG)
Ram Rao (Digital UEG)
David Rosenthal (Sun)
Dave Winchell (Digital UEG)

The implementors of initial server who provided useful input are:

Susan Angebranndt (Digital)
Raymond Drewry (Digital)
Todd Newman (Digital)

The invited reviewers who provided useful input are:

Andrew Cherenson (Berkeley)
Burns Fisher (Digital)
Dan Garfinkel (HP)
Leo Hourvitz (Next)
Brock Krizan (HP)
David Laidlaw (Stellar)
Dave Mellinger (Interleaf)
Ron Newman (MIT)
John Ousterhout (Berkeley)
Andrew Palay (ITC CMU)
Ralph Swick (MIT)
Craig Taylor (Sun)
Jeffery Vroom (Stellar)

Thanks go to Al Mento of Digital's UEG Documentation Group for formatting this document.

This document does not attempt to provide the rationale or pragmatics required to fully understand the protocol or to place it in perspective within a complete system.

The protocol contains many management mechanisms that are not intended for normal applications. Not all mechanisms are needed to build a particular user interface. It is important to keep in mind that the protocol is intended to provide mechanism, not policy.

Robert W. Scheifler
Massachusetts Institute of Technology
Laboratory for Computer Science

1. Terminology

Access control list

X maintains a list of hosts from which client programs may be run. By default, only programs on the local host may use the display, plus any hosts specified in an initial list read by the server. This ''access control list'' can be changed by clients on the local host. Some server implementations may also implement other authorization mechanisms in addition to or in place of this mechanism. The action of this mechanism may be conditional based on the authorization protocol name and data received by the server at connection setup.

Active grab

A grab is ''active'' when the pointer or keyboard is actually owned by the single grabbing client.

Ancestors

If W is an inferior of A, then A is an ''ancestor'' of W.

Atom

An ''atom'' is a unique ID corresponding to a string name. Atoms are used to identify properties, types, and selections.

Background

An InputOutput window can have a ''background'', defined as a pixmap, in which case when regions of the window have their contents lost or invalidated, the server will automatically tile those regions with the background.

Backing store

When a server maintains the contents of a window, the off-screen saved pixels are known as a ''backing store''.

Bit gravity

When a window is resized, the contents of the window are not necessarily discarded. It is possible to request the server (though no guarantees are made) to relocate the previous contents to some region of the window. This attraction of window contents for some location of a window is known as ''bit gravity''.

Bit plane

When a pixmap or window is thought of as a stack of bitmaps, each bitmap is called a ''bit plane'' or ''plane''.

Bitmap

A ''bitmap'' is a pixmap of depth one.

Border

An InputOutput window can have a ''border'', with equal thickness on all four sides of the window. The contents of the border are defined by a pixmap, and the server automatically maintains the contents of the border. Exposure events are never generated for border regions.

Button grabbing

Buttons on the pointer may be passively ''grabbed'' by a client. When the button is pressed, the pointer is then actively grabbed by the client.

Byte order

For image (pixmap/bitmap) data, byte order is defined by the server, and clients with different native byte ordering must swap bytes as necessary. For all other parts of the protocol, the byte order is defined by the client, and the server swaps bytes as necessary.

Children

The ''children'' of a window are its first-level subwindows.

Client

An application program connects to the window system server by some interprocess communication (IPC) path, such as a TCP connection or a shared memory buffer. This program is referred to as a ''client'' of the window system server. More precisely, the client is the IPC path itself; a program with multiple paths open to the server is viewed as multiple clients by the protocol. Resource lifetimes are controlled by connection lifetimes, not by program lifetimes.

Clipping region

In a graphics context, a bitmap or list of rectangles can be specified to restrict output to a particular region of the window. The image defined by the bitmap or rectangles is called a ''clipping region''.

Colormap

A ''colormap'' consists of a set of entries defining color values. The colormap associated with a window is used to display the contents of the window; each pixel value indexes the colormap to produce RGB values that drive the guns of a monitor. Depending on hardware limitations, one or more colormaps may be installed at one time, such that windows associated with those maps display with correct colors.

Connection

The IPC path between the server and client program is known as a ''connection''. A client program typically (but not necessarily) has one connection to the server over which requests and events are sent.

Containment

A window ''contains'' the pointer if the window is viewable and the hotspot of the cursor is within a visible region of the window or a visible region of one of its inferiors. The border of the window is included as part of the window for containment. The pointer is ''in'' a window if the window contains the pointer but no inferior contains the pointer.

Coordinate system

The coordinate system has X horizontal and Y vertical, with the origin [0, 0] at the upper left. Coordinates are discrete and are in terms of pixels. Each window and pixmap has its own coordinate system. For a window, the origin is at the inside upper-left, inside the border.

Cursor

A ''cursor'' is the visible shape of the pointer on a screen. It consists of a hot spot, a source bitmap, a shape bitmap, and a pair of colors. The cursor defined for a window controls the visible appearance when the pointer is in that window.

Depth

The ''depth'' of a window or pixmap is number of bits per pixel it has. The depth of a graphics context is the depth of the drawables it can be used in conjunction with for graphics output.

Device

Keyboards, mice, tablets, track-balls, button boxes, and so on are all collectively known as input ''devices''. The core protocol only deals with two devices, ''the keyboard'' and ''the pointer''.

DirectColor

A class of colormap in which a pixel value is decomposed into three separate subfields for indexing. The first subfield indexes an array to produce red intensity values. The second indexes a second array to produce blue intensity values. The third subfield indexes a third array to produce green intensity values. The RGB values can be changed dynamically.

Display

A server, together with its screens and input devices, is called a "display".

Drawable

Both windows and pixmaps may be used as sources and destinations in graphics operations. These are collectively known as "drawables". However, an InputOnly window cannot be used as a source or destination in a graphics operation.

Event

Clients are informed of information asynchronously by means of "events". These events can be either asynchronously generated from devices or can generated as side effects of client requests. Events are grouped into types; events are never sent to a client by the server unless the client has specificially asked to be informed of that type of event, but other clients can force events to be sent to other clients. Events are typically reported relative to a window.

Event mask

Events are requested relative to a window. The set of event types a client requests relative to a window described using an "event mask".

Event sychronization

There are certain race conditions possible when demultiplexing device events to clients (in particular deciding where pointer and keyboard events should be sent when in the middle of window management operations). The event synchronization mechanism allows synchronous processing of device events.

Event propagation

Device-related events "propagate" from the source window to ancestor windows until some client has expressed interest in handling that type of event or until the event is discarded explicitly.

Event source

The smallest window containing the pointer is the "source" of a device related event.

Exposure event

Servers do not guarantee to preserve the contents of windows when windows are obscured or reconfigured. Exposure events are sent to clients to inform them when contents of regions of windows have been lost.

Extension

Named "extensions" to the core protocol can be defined to extend the system. Extension to output requests, resources, and event types are all possible and are expected.

Focus window

The "focus window" is another term for the input focus.

Font

A "font" is a matrix of glyphs (typically characters). The protocol does no translation or interpretation of character sets. The client simply indicates values used to index the glyph array. A font contains additional metric information to determine inter-glyph and inter-line spacing.

Glyph

A "glyph" is an image, typically of a character, in a font.

Grab

Keyboard keys, the keyboard, pointer buttons, the pointer, and the server can be "grabbed" for exclusive use by a client. In general, these facilities are not intended to be used by normal applications but are intended for various input and window managers to implement various styles of user interfaces.

Gravity

See ''bit gravity'' and ''window gravity''.

GrayScale

GrayScale can be viewed as a degenerate case of PseudoColor, in which the red, green, and blue values in any given colormap entry are equal, thus producing shades of gray. The gray values can be changed dynamically.

GC, GContext

Shorthand for ''graphics context''.

Graphics context

Various information for graphics output is stored in a ''graphics context'' (or ''GC'' or ''gcontext'') such as foreground pixel, background pixel, line width, clipping region, and so on. A graphics context can only be used with drawables that have the same root and the same depth as the graphics context.

Hotspot

A cursor has an associated ''hot spot'' that defines a point in the cursor that corresponds to the coordinates reported for the pointer.

Identifier

Each resource has an ''identifier'', a unique value associated with it that clients use to name the resource. An identifier can be used over any connection to name the resource.

Inferiors

The ''inferiors'' of a window are all of the subwindows nested below it: the children, the children's children, and so on.

Input focus

The ''input focus'' is normally a window defining the scope for processing of keyboard input. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported normally. Otherwise, the event is reported with respect to the focus window. The input focus also can be set such that all keyboard events are discarded and such that the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event.

Input manager

Control over keyboard input is typically provided by an ''input manager'' client.

InputOnly window

A window that cannot be used for graphics requests. InputOnly windows are ''invisible'' and can be used to control such things as cursors, input event generation, and grabbing. InputOnly windows cannot have InputOutput windows as inferiors.

InputOutput window

The ''normal'' kind of opaque window, used for both input and output. InputOutput windows can have both InputOutput and InputOnly windows as inferiors.

Key grabbing

Keys on the keyboard may be passively ''grabbed'' by a client. When the key is pressed, the keyboard is then actively grabbed by the client.

Keyboard grabbing

A client can actively ''grab'' control of the keyboard, and key events will be sent to that client rather than the client the events would normally have been sent to.

Keysym

An encoding of a symbol on a keycap on a keyboard.

Mapped

A window is said to be ''mapped'' if a map call has been performed on it. Unmapped windows and their inferiors are never viewable or visible.

Modifier keys

Shift, Control, Meta, Super, Hyper, ALT, Compose, Apple, CapsLock, ShiftLock, and similar keys are called ''modifier'' keys.

Monochrome

A special case of StaticGray, in which there are only two colormap entries.

Obscure

A window is ''obscured'' if some other window ''obscures'' it. Window A ''obscures'' window B if both are viewable InputOutput windows and A is higher in the global stacking order, and the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Note the (fine) distinction with ''occludes''. Also note that window borders are included in the calculation. Also note that a window can be obscured and yet still have visible regions.

Occlude

A window is ''occluded'' if some other window ''occludes'' it. Window A ''occludes'' window B if both are mapped and A is higher in the global stacking order, and the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Note the (fine) distinction with ''obscures''. Also note that window borders are included in the calculation.

Padding

Some padding bytes are inserted in the data stream to maintain alignment of the protocol requests on natural boundaries. This increases ease of portability to some machine architectures.

Parent window

If C is a child of P, then P is the ''parent'' of C.

Passive grab

Grabbing a key or button is a ''passive'' grab. The grab activates when the key or button is actually pressed.

Pixel value

A ''pixel'' is an N-bit value, where N is the number of bit planes used in (that is, the depth of) a particular window or pixmap. For a window, a pixel value indexes a colormap to derive an actual color to be displayed.

Pixmap

A ''pixmap'' is a three dimensional array of bits. A pixmap is normally thought of as a two dimensional array of pixels, where each pixel can be a value from 0 to (2^N)-1 and where N is the depth (z axis) of the pixmap. A pixmap can also be thought of as a stack of N bitmaps.

Plane

When a pixmap or window is thought of as a stack of bitmaps, each bitmap is called a ''plane'' or ''bit plane''.

Plane mask

Graphics operations can be restricted to only affect a subset of bit planes of a destination. A ''plane mask'' is a bit mask describing which planes are to be modified and is stored in a graphics context.

Pointer

The ''pointer'' is the pointing device attached to the cursor and tracked on the screens.

Pointer grabbing

A client can actively ''grab'' control of the pointer, and button and motion events will be sent to that client rather than the client the events would normally have been sent to.

Pointing device

A ''pointing device'' is typically a mouse, tablet, or some other device with effective dimensional motion. There is only one visible cursor is defined by the core protocol, and it tracks whatever pointing device is attached as the pointer.

Property

Windows may have associated ''properties'', consisting of a name, a type, a data format, and some data. The protocol places no interpretation on properties. They are intended as a general-purpose naming mechanism for clients. For example, clients might share information such as resize hints, program names, and icon formats with a window manager by means of properties.

Property list

The ''property list'' of a window is the list of properties that have been defined for the window.

PseudoColor

A class of colormap in which a pixel value indexes the colormap to produce independent red, green, and blue values. That is, the colormap is viewed as an array of triples (RGB values). The RGB values can be changed dynamically.

Redirecting control

Window managers (or client programs) may wish to enforce window layout policy in various ways. When a client attempts to change the size or position of a window, the operation may be ''redirected'' to a specified client, rather than the operation actually being performed.

Reply

Information requested by a client program is sent back to the client with a ''reply''. Both events and replies are multiplexed on the same connection. Most requests do not generate replies. Some requests generate multiple replies.

Request

A command to the server is called a ''request''. It is a single block of data sent over a connection.

Resource

Windows, pixmaps, cursors, fonts, graphics contexts, and colormaps are known as ''resources''. They all have unique identifiers associated with them for naming purposes. The lifetime of a resource is bounded by the lifetime of the connection over which the resource was created.

RGB values

''Red, Green, and Blue'' intensity values are used to define color. These values are always represented as 16-bit unsigned numbers, with zero being minimum intensity and 65535 being the maximum intensity. The values are scaled by the server to match the display hardware.

Root

The ''root'' of a pixmap or graphics context is the same as the root of whatever drawable was used when the pixmap or graphics context was created. The ''root'' of a window is the root window under which the window was created.

Root window

Each screen has a ''root window'' covering it. It cannot be reconfigured or unmapped, but it otherwise acts as a full fledged window. A root window has no parent.

Save set

The ''save set'' of a client is a list of other client's windows that, if they are inferiors of one of the client's windows at connection close, should not be destroyed and that should be remapped if it is unmapped. Save sets are typically used by window managers to avoid lost windows if the manager should terminate abnormally.

Scanline

A ''scanline'' is a list of pixel or bit values viewed as a horizontal row (all values having the same y coordinate) of an image, with the values ordered by increasing x coordinate.

Scanline order

An image represented in ''scanline order'' contains scanlines ordered by increasing y coordinate.

Screen

A server may provide several independent ''screens'', which typically have physically independent monitors. This would be the expected configuration when there is only a single keyboard and pointer shared among the screens.

Selection

A ''selection'' can be thought of as an indirect property with dynamic type. That is, rather than having the property stored in the server, it is maintained by some client (the ''owner''). A selection is global in nature and is thought of as belonging to the user (but maintained by clients), rather than as being private to a particular window subhierarchy or a particular set of clients. When a client asks for the contents of a selection, it specifies a selection ''target type''. This target type can be used to control the transmitted representation of the contents. For example, if the selection is ''the last thing the user clicked on'', and that is currently an image, then the target type might specify whether the contents of the image should be sent in XYFormat or ZFormat. The target type can also be used to control the class of contents transmitted; for example, asking for the ''looks'' (fonts, line spacing, indentation, and so on) of a paragraph selection, rather than the text of the paragraph. The target type can also be used for other purposes. The semantics is not constrained by the protocol.

Server

The ''server'' provides the basic windowing mechanism. It handles IPC connections from clients, demultiplexes graphics requests onto the screens, and multiplexes input back to the appropriate clients.

Server grabbing

The server can be ''grabbed'' by a single client for exclusive use. This prevents processing of any requests from other client connections until the grab is complete. This is typically only a transient state for such things as rubber-banding and pop-up menus, or to execute requests indivisibly.

Sibling

Children of the same parent window are known as ''sibling'' windows.

StaticColor

StaticColor can be viewed as a degenerate case of PseudoColor, in which the RGB values are predefined and read-only.

StaticGray

StaticGray can be viewed as a degenerate case of GrayScale, in which the gray values are predefined and read-only. The values are typically (near-)linear increasing ramps.

Stacking order

Sibling windows may "stack" on top of each other. Windows above both obscure and occlude lower windows. This is similar to paper on a desk. The relationship between sibling windows is known as the "stacking order".

Stipple

A "stipple pattern" is a bitmap that is used to tile a region to serve as an additional clip mask for a fill operation with the foreground color.

Tile

A pixmap can be replicated in two dimensions to "tile" a region. The pixmap itself is also known as a "tile".

Timestamp

A time value, expressed in milliseconds, typically since the last server reset. Timestamp values wrap around (after about 49.7 days). The server, given its current time is represented by timestamp T, always interprets timestamps from clients by treating half of the timestamp space as being earlier in time than T and half of the timestamp space as being later in time than T. One timestamp value (named CurrentTime) is never generated by the server. This value is reserved for use in requests to represent the current server time.

TrueColor

TrueColor can be viewed as a degenerate case of DirectColor, in which the subfields in the pixel value directly encode the corresponding RGB values. That is, the colormap has predefined read-only RGB values. The values are typically (near-)linear increasing ramps.

Type

A type is an arbitrary atom used to identify the interpretation of property data. Types are completely uninterpreted by the server and are solely for the benefit of clients.

Viewable

A window is "viewable" if it and all of its ancestors are mapped. This does not imply that any portion of the window is actually visible. Graphics requests can be performed on a window when it is not viewable, but output will not be retained unless the server is maintaining backing store.

Visible

A region of a window is "visible" if someone looking at the screen can actually "see" it: the window is viewable and the region is not occluded by any other window.

Window gravity

When windows are resized, subwindows may be repositioned automatically relative to some position in the window. This attraction of a subwindow to some part of its parent is known as "window gravity".

Window manager

Manipulation of windows on the screen and much of the user interface (policy) is typically provided by a "window manager" client.

XYFormat

The data for a pixmap is said to be in "XYFormat" if it is organized as a set of bitmaps representing individual bit planes, with the planes appearing from most to least significant in bit order.

ZFormat

The data for a pixmap is said to be in ''ZFormat'' if it is organized as a set of pixel values in scanline order.

2. Protocol Formats

Request Format

Every request contains an 8-bit ''major'' opcode, and a 16-bit length field expressed in units of 4 bytes. Every request consists of 4 bytes of header (containing the major opcode, the length field, and a data byte) followed by zero or more additional bytes of data. The length field defines the total length of the request, including the header. The length field in a request must equal the minimum length required to contain the request. If the specified length is smaller or larger than the required length, an error is generated. Unused bytes in a request are not required to be zero. Major opcodes 128 through 255 are reserved for extensions. Extensions are intended to contain multiple requests, so extension requests typically have an additional minor opcode encoded in the ''spare'' data byte in the request header, but the placement and interpretation of this minor opcode, and all other fields in extension requests, are not defined by the core protocol. Every request on a given connection is implicitly assigned a sequence number, starting with one, that is used in replies, errors, and events.

Reply Format

Every reply contains a 32-bit length field expressed in units of 4 bytes. Every reply consists of 32 bytes, followed by zero or more additional bytes of data, as specified in the length field. Unused bytes within a reply are not guaranteed to be zero. Every reply also contains the least significant 16 bits of the sequence number of the corresponding request.

Error Format

Error reports are 32 bytes long. Every error includes an 8-bit error code. Error codes 128 through 255 are reserved for extensions. Every error also includes the major and minor opcodes of the failed request, and the least significant 16 bits of the sequence number of the request. For the following errors (see Section 5) the failing resource ID is also returned: Colormap, Cursor, Drawable, Font, GContext, IDChoice, Pixmap, and Window. For Atom errors, the failing atom is returned. For Value errors, the failing value is returned. Other core errors return no additional data. Unused bytes within an error are not guaranteed to be zero.

Event Format

Events are 32 bytes long. Unused bytes within an event are not guaranteed to be zero. Every event contains an 8-bit type code. The most significant bit in this code is set if the event was generated from a SendEvent request. Event codes 64 through 127 are reserved for extensions, although the core protocol does not define a mechanism for selecting interest in such events. Every core event (with the exception of KeymapNotify) also contains the least significant 16 bits of the sequence number of the last request issued by the client that was (or is currently being) processed by the server.

3. Syntax

The syntax {...} encloses a set of alternatives.

The syntax [...] encloses a set of structure components.

In general, TYPEs are in upper-case and AlternativeValues are capitalized.

Requests in Section 10 are described in the following format:

RequestName
arg1: type1
...
argN: typeN
=>
result1: type1
...
resultM: typeM

Errors: kind1, ..., kindK

Description.

If no => is present in the description, then the request has no reply (it is asynchronous), although errors may still be reported. If =>+ is used, then one or more replies can be generated for a single request.

Events in Section 12 are described in the following format:

EventName
value1: type1
...
valueN: typeN

Description.

4. Common Types

LISTofFOO

A type name of the form LISTofFOO means a counted list of elements of type FOO. The size of the length field may vary (it is not necessarily the same size as a FOO), in some cases may be implicit, and is not fully specified in this document. Except where explicitly noted, zero-length lists are legal.

BITMASK
LISTofVALUE

The types BITMASK and LISTofVALUE are somewhat special. Various requests contain arguments of the form:

value-mask: BITMASK
value-list: LISTofVALUE

used to allow the client to specify a subset of a heterogeneous collection of "optional" arguments. The value-mask specifies which arguments are to be provided; each such argument is assigned a unique bit position. The representation of the BITMASK will typically contain more bits than there are defined arguments. The unused bits in the value-mask must be zero (or the server generates a Value error). The value-list contains one value for each one bit in the mask, from least to most significant bit in the mask. Each value is represented with 4 bytes, but the actual value occupies only the least significant bytes as required. The values of the unused bytes do not matter.

OR

A type of the form "T1 or ... or Tn" means the union of the indicated types. A single-element type is given as the element without enclosing braces.

WINDOW: 32-bit value (top 3 bits guaranteed to be zero)

PIXMAP: 32-bit value (top 3 bits guaranteed to be zero)
CURSOR: 32-bit value (top 3 bits guaranteed to be zero)
FONT: 32-bit value (top 3 bits guaranteed to be zero)
GCONTEXT: 32-bit value (top 3 bits guaranteed to be zero)
COLORMAP: 32-bit value (top 3 bits guaranteed to be zero)
DRAWABLE: WINDOW or PIXMAP
FONTABLE: FONT or GCONTEXT
ATOM: 32-bit value (top 3 bits guaranteed to be zero)
VISUALID: 32-bit value (top 3 bits guaranteed to be zero)
VALUE: 32-bit quantity (used only in LISTofVALUE)
BYTE: 8-bit value
INT8: 8-bit signed integer
INT16: 16-bit signed integer
INT32: 32-bit signed integer
CARD8: 8-bit unsigned integer
CARD16: 16-bit unsigned integer
CARD32: 32-bit unsigned integer
TIMESTAMP: CARD32

BITGRAVITY: {Forget, Static, NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}

WINGRAVITY: {Unmap, Static, NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}

BOOL: {True, False}

EVENT: {KeyPress, KeyRelease, OwnerGrabButton, ButtonPress, ButtonRelease, EnterWindow, LeaveWindow, PointerMotion, PointerMotionHint, Button1Motion, Button2Motion, Button3Motion, Button4Motion, Button5Motion, ButtonMotion, Exposure, VisibilityChange, StructureNotify, ResizeRedirect, SubstructureNotify, SubstructureRedirect, FocusChange, PropertyChange, ColormapChange, KeymapState}

POINTEREVENT: {ButtonPress, ButtonRelease, EnterWindow, LeaveWindow, PointerMotion, PointerMotionHint, Button1Motion, Button2Motion, Button3Motion, Button4Motion, Button5Motion, ButtonMotion, KeymapState}

DEVICEEVENT: {KeyPress, KeyRelease, ButtonPress, ButtonRelease, PointerMotion, Button1Motion, Button2Motion, Button3Motion, Button4Motion, Button5Motion, ButtonMotion}

KEYSYM: 32-bit value (top 3 bits guaranteed to be zero)
KEYCODE: CARD8
BUTTON: CARD8
KEYMASK: {Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, Mod5}

BUTMASK: { Button1, Button2, Button3, Button4, Button5 }

KEYBUTMASK: KEYMASK or BUTMASK

STRING8: LISTofCARD8

STRING16: LISTofCHAR2B

CHAR2B: [byte1, byte2: CARD8]

POINT: [x, y: INT16]

RECTANGLE: [x, y: INT16,
width, height: CARD16]

ARC: [x, y: INT16,
width, height: CARD16,
angle1, angle2: INT16]

HOST: [family: { Internet, DECnet, Chaos }
address: LISTofBYTE]

The [x,y] coordinates of a RECTANGLE specify the upper-left corner.

The primary interpretation of "large" characters in a STRING16 is that they are composed of 2-bytes used to index a 2-D matrix. Hence, the use of CHAR2B rather than CARD16. This corresponds to the JIS/ISO method of indexing 2-byte characters. It is expected that most "large" fonts will be defined with 2-byte matrix indexing. For large fonts constructed with linear indexing, a CHAR2B can be interpreted as a 16-bit number by treating byte1 as the most significant byte. This means that clients should always transmit such 16-bit character values most significant byte first, as the server will never byte-swap CHAR2B quantities.

The length, format, and interpretation of a HOST address are specific to the family; see ChangeHosts.

5. Errors

In general, when a request terminates with an error, the request has no side effects (that is, there is no partial execution). The only requests for which this is not true are ChangeWindowAttributes, ChangeGC, PolyText8, PolyText16, FreeColors, StoreColors, and ChangeKeyboardControl.

The following error codes can be returned by the various requests:

Access

An attempt to grab a key/button combination already grabbed by another client.

An attempt to free a colormap entry not allocated by the client.

An attempt to store into a read-only or an unallocated colormap entry.

An attempt to modify the access control list from other than the local host (or otherwise authorized client).

An attempt to select an event type, that at most one client can select at a time, when another client has already selected it.

Alloc

The server failed to allocate the requested resource.

Note that this only covers allocation errors at a very coarse level and is not intended to (nor can it in practice hope to) cover all cases of a server running out of allocation space in the middle of service. The semantics when a server runs out of allocation space are left unspecified.

Atom

A value for an ATOM argument does not name a defined ATOM.

Colormap

A value for a COLORMAP argument does not name a defined COLORMAP.

Cursor

A value for a CURSOR argument does not name a defined CURSOR.

Drawable

A value for a DRAWABLE argument does not name a defined WINDOW or PIXMAP.

Font

A value for a FONT argument does not name a defined FONT.

A value for a FONTABLE argument does not name a defined FONT or a defined GCONTEXT.

GContext

A value for a GCONTEXT argument does not name a defined GCONTEXT.

IDChoice

The value chosen for a resource identifier either is not included in the range assigned to the client or is already in use.

Implementation

The server does not implement some aspect of the request. A server that generates this error for a core request is deficient. As such, this error is not listed for any of the requests, but clients should be prepared to receive such errors and handle or discard them.

Length

The length of a request is shorter or longer than that required to minimally contain the arguments.

The length of a request exceeds the maximum length accepted by the server.

Match

An InputOnly window is used as a DRAWABLE.

In a graphics request, the GCONTEXT argument does not have the same root and depth as the destination DRAWABLE argument.

Some argument (or pair of arguments) has the correct type and range, but it fails to "match" in some other way required by the request.

Name

A font or color of the specified name does not exist.

Pixmap

A value for a PIXMAP argument does not name a defined PIXMAP.

Request

The major or minor opcode does not specify a valid request.

Value

Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives typically can generate this error (due to the encoding).

Window

A value for a WINDOW argument does not name a defined WINDOW.

Note

The Atom, Colormap, Cursor, Drawable, Font, GContext, Pixmap, and Window errors are also used when the argument type is extended by union with a set of fixed alternatives, for example, <WINDOW or PointerRoot or None>.

6. Keyboards

A KEYCODE represents a physical (or logical) key. Keycodes lie in the inclusive range [8,255]. A keycode value carries no intrinsic information, although server implementors may attempt to encode geometry (for example, matrix) information in some fashion, to be interpreted in a server-dependent fashion. The mapping between keys and keycodes cannot be changed by means of the protocol.

A KEYSYM is an encoding of a symbol on the cap of a key. The set of defined KEYSYMs include the character sets Latin 1, Latin 2, Latin 3, Latin 4, Kana, Arabic, Cryllic, Greek, Tech, Special, Publish, APL, and Hebrew, plus a set of symbols common on keyboards (RETURN, HELP, TAB, and so on). KEYSYMs with the most significant bit (of the 29 bits) set are reserved as "vendor-specific".

A list of KEYSYMs is associated with each KEYCODE; the length of the list can vary with each KEYCODE. The list is intended to convey the set of symbols on the corresponding key. By convention, if the list contains a single KEYSYM, and that KEYSYM is alphabetic and case distinction is relevant for it, then it should be treated as equivalent to a two-element list of the lowercase and uppercase KEYSYMs. For example, if the list contains the single KEYSYM for uppercase A, then the client should treat it as if it were instead a pair with lowercase "a" as the first KEYSYM and uppercase "A" as the second KEYSYM.

For any KEYCODE, the first KEYSYM in the list normally should be chosen as the interpretation of a KeyPress when no modifier keys are down. The second KEYSYM in the list normally should be chosen when the Shift modifier is on or when the Lock modifier is on and Lock is interpreted as ShiftLock. When the Lock modifier is on and is interpreted as CapsLock, it is suggested that the Shift modifier first be applied to choose a KEYSYM, but if that KEYSYM is lowercase alphabetic, the corresponding uppercase KEYSYM should be used instead. Other interpretations of CapsLock are possible. For example, it may be viewed as equivalent to ShiftLock, but only applying when the first KEYSYM is lowercase alphabetic and the second KEYSYM is the corresponding uppercase alphabetic. No interpretation of KEYSYMs beyond the first two in a list is suggested here. No spatial geometry of the symbols on the key is defined by their order in the KEYSYM list, although a geometry might be defined on a vendor-specific basis.

The mapping between KEYCODEs and KEYSYMs is not used directly by the server; it is merely stored for reading and writing by clients.

The KEYMASK modifier named Lock is intended to be mapped to either a CapsLock or a ShiftLock key, but which one is left as application and/or user specific. However, it is suggested that the determination be made according to the associated KEYSYM(s) of the corresponding KEYCODE.

7. Pointers

Buttons are always numbered starting with one.

8. Predefined Atoms

Predefined atoms are not strictly necessary and may not be useful in all environments, but they will eliminate many InternAtom requests in most applications. Note that "predefined" is only in the sense of having numeric values, not in the sense of having required semantics. The core protocol imposes no semantics on these names, except as they are used in FONTPROP structures (see QueryFont).

The following names have predefined atom values. Note that upper/lower case matters.

ARC	ITALIC_ANGLE	STRING
ATOM	MAX_SPACE	SUBSCRIPT_X
BITMAP	MIN_SPACE	SUBSCRIPT_Y
CAP_HEIGHT	NORM_SPACE	SUPERSCRIPT_X
CARDINAL	NOTICE	SUPERSCRIPT_Y
COLORMAP	PIXMAP	UNDERLINE_POSITION
COPYRIGHT	POINT	UNDERLINE_THICKNESS
CURSOR	POINT_SIZE	VISUALID
CUT_BUFFER0	PRIMARY	WEIGHT
CUT_BUFFER1	QUAD_WIDTH	WINDOW
CUT_BUFFER2	RECTANGLE	WM_CLASS
CUT_BUFFER3	RESOLUTION	WM_CLIENT_MACHINE
CUT_BUFFER4	RESOURCE_MANAGER	WM_COMMAND
CUT_BUFFER5	RGB_BEST_MAP	WM_HINTS
CUT_BUFFER6	RGB_BLUE_MAP	WM_ICON_NAME
CUT_BUFFER7	RGB_COLOR_MAP	WM_ICON_SIZE
DRAWABLE	RGB_DEFAULT_MAP	WM_NAME
END_SPACE	RGB_GRAY_MAP	WM_NORMAL_HINTS
FAMILY_NAME	RGB_GREEN_MAP	WM_SIZE_HINTS
FONT	RGB_RED_MAP	WM_TRANSIENT_FOR
FONT_NAME	SECONDARY	WM_ZOOM_HINTS
FULL_NAME	STRIKEOUT_ASCENT	X_HEIGHT
INTEGER		STRIKEOUT_DESCENT

To avoid conflicts with possible future names for which semantics might be imposed (either at the protocol level or in terms of higher level user interface models), names beginning with an underscore should be used for atoms that are private to a particular vendor or organization. To guarantee no conflicts between vendors and organizations, additional prefixes need to be used, but the mechanism for choosing such prefixes is not defined here. For names private to a single application or end user but stored in "globally accessible" locations, it is suggested that two leading underscores be used to avoid conflicts with other names.

9. Connection Setup

For remote clients, the X protocol can be built on top of any reliable byte stream.

The client must send an initial byte of data to identify the byte order to be employed. The value of the byte must be octal 102 or 154. The value 102 (ASCII uppercase B) means values are transmitted most significant byte first, and value 154 (ASCII lowercase l) means values are transmitted least significant byte first. Except where explicitly noted in the protocol, all 16-bit and 32-bit quantities sent by the client must be transmitted with this byte order, and all 16-bit and 32-bit quantities returned by the server will be transmitted with this byte order.

Following the byte-order byte, the following information is sent by the client at connection setup:

protocol-major-version: CARD16
protocol-minor-version: CARD16
authorization-protocol-name: STRING8
authorization-protocol-data: STRING8

The version numbers indicate what version of the protocol the client expects the server to implement. See below for an explanation.

The authorization name indicates what authorization protocol the client expects the server to use, and the data is specific to that protocol. Specification of valid authorization mechanisms is not part of the core X protocol. It is hoped that eventually one authorization protocol will be agreed upon. In the mean time, a server that implements a different protocol than the client expects or that only implements the host-based mechanism may simply ignore this information. If both name and data strings are empty, this is to be interpreted as ''no explicit authorization''.

Received by the client at connection setup:

success: BOOL
protocol-major-version: CARD16
protocol-minor-version: CARD16
length: CARD16

Length is the amount of additional data to follow, in units of 4 bytes. The version numbers are an escape hatch in case future revisions of the protocol are necessary. In general, the major version would increment for incompatible changes, and the minor version would increment for small upward compatible changes. Barring changes, the major version will be eleven, and the minor version will be zero. The protocol version numbers returned indicate the protocol the server actually supports. This might not equal the version sent by the client. The server can (but need not) refuse connections from clients that offer a different version than the server supports. A server can (but need not) support more than one version simultaneously.

Additional data received if authorization fails:

reason: STRING8

Additional data received if authorization is accepted:

vendor: STRING8
release-number: CARD32
resource-id-base, resource-id-mask: CARD32
image-byte-order: {LSBFirst, MSBFirst}
bitmap-scanline-unit: {8, 16, 32}
bitmap-scanline-pad: {8, 16, 32}
bitmap-bit-order: {LeastSignificant, MostSignificant}
pixmap-formats: LISTofFORMAT
roots: LISTofSCREEN
motion-buffer-size: CARD32
maximum-request-length: CARD16
min-keycode, max-keycode: KEYCODE

where

FORMAT:	[depth: CARD8, bits-per-pixel: {1, 4, 8, 16, 24, 32} scanline-pad: {8, 16, 32}]

SCREEN:	[root: WINDOW width-in-pixels, height-in-pixels: CARD16 width-in-millimeters, height-in-millimeters: CARD16 allowed-depths: LISTofDEPTH root-depth: CARD8 root-visual: VISUALID default-colormap: COLORMAP white-pixel, black-pixel: CARD32 min-installed-maps, max-installed-maps: CARD16 backing-stores: {Never, WhenMapped, Always} save-unders: BOOL current-input-masks: SETofEVENT]
DEPTH:	[depth: CARD8 visuals: LISTofVISUALTYPE]
VISUALTYPE:	[visual-id: VISUALID class: {StaticGray, StaticColor, TrueColor, GrayScale, PseudoColor, DirectColor} red-mask, green-mask, blue-mask: CARD32 bits-per-rgb-value: CARD8 colormap-entries: CARD16]

Per server information:

The vendor string gives some indentification of the owner of the server implementation. The semantics of the release-number is controlled by the vendor.

The resource-id-mask contains a single contiguous set of bits (at least 18). The client allocates resource IDs for types WINDOW, PIXMAP, CURSOR, FONT, GCONTEXT, and COLORMAP by choosing a value with (only) some subset of these bits set, and ORing it with resource-id-base. Only values constructed in this way can be used to name newly created resources over this connection. Resource IDs never have the top 3 bits set. The client is not restricted to linear or contiguous allocation of resource IDs. Once an ID has been freed, it can be reused, but this should not be necessary. An ID must be unique with respect to the IDs of all other resources, not just other resources of the same type. However, note that the value spaces of a) resource identifiers, b) atoms, c) visualids, and d) keysyms are distinguished by context, and as such are not required to be disjoint (for example, a given numeric value might be both a valid window ID, a valid atom, and a valid keysym).

Although the server is in general responsible for byte swapping data to match the client, images are always transmitted and received in formats (including byte order) specified by the server. The byte order for images is given by image-byte-order and applies to each scanline unit in XYFormat (bitmap) format and to each pixel value in ZFormat.

A bitmap is represented in scanline order. Each scanline is padded to a multiple of bits as given by bitmap-scanline-pad. The pad bits are of arbitrary value. The scanline is quantized in multiples of bits as given by bitmap-scanline-unit. The bitmap-scanline-unit is always less than or equal to the bitmap-scanline-pad. Within each unit, the leftmost bit in the bitmap is either the least or most significant bit in the unit, as given by bitmap-bit-order. If a pixmap is represented in XYFormat, each plane is represented as a bitmap, and the planes appear from most to least significant in bit order, with no padding between planes.

Pixmap-formats contains one entry for each depth value. The entry describes the ZFormat used to represent images of that depth. An entry for a depth is included if any screen supports that depth, and all screens supporting that depth must support (only) that ZFormat for that depth. In ZFormat, the pixels are in scanline order, left to right within a scanline. The number of bits used to hold each pixel is given by bits-per-pixel. Bits-per-pixel may be larger than strictly required by the depth, in which case the least significant bits are used to hold the pixmap data, and the values of the unused high order bits are undefined. When the bits-per-pixel is 4, the order of nibbles in the byte is the same as the image byte-order. When the bits-per-pixel is 1, the format is identical for bitmap format. Each scanline is padded to a multiple of bits as given by scanline-pad. When bits-per-pixel is 1, this will be identical to bitmap-scanline-pad.

How a pointing device roams the screens is up to the server implementation and is transparent to the protocol. No geometry among screens is defined.

The server may retain the recent history of pointer motion and to a finer granularity than is reported by MotionNotify events. Such history is available by means of the GetMotionEvents request. The approximate size of the history buffer is given by motion-buffer-size.

Maximum-request-length specifies the maximum length of a request, in 4-byte units, accepted by the server. That is, this is the maximum value that can appear in the length field of a request. Requests larger than this generate a Length error, and the server will read and simply discard the entire request. Maximum-request-length will always be at least 4096 (that is, requests of length up to and including 16384 bytes will be accepted by all servers).

Min-keycode and max-keycode specify the smallest and largest keycode values transmitted by the server. Min-keycode is never less than 8, and max-keycode is never greater than 255. Not all keycodes in this range are required to have corresponding keys.

Per screen information:

The allowed-depths specifies what pixmap and window depths are supported. Pixmaps are supported for each depth listed, and windows of that depth are supported if at least one visual type is listed for the depth. A pixmap depth of one is always supported and listed, but windows of depth one might not be supported. A depth of zero is never listed, but zero-depth InputOnly windows are always supported.

Root-depth and root-visual specify the depth and visual type of the root window. Width-in-pixels and height-in-pixels specify the size of the root window (which cannot be changed). The class of the root window is always InputOutput. Width-in-millimeters and height-in-millimeters can be used to determine the physical size and the aspect ratio.

The default-colormap is the one initially associated with the root window. Clients with minimal color requirements creating windows of the same depth as the root may want to allocate from this map by default.

Black-pixel and white-pixel can be used in implementing a "monochrome" application. These pixel values are for permanently allocated entries in the default-colormap. The actual RGB values may be settable on some screens and, in any case, may not actual be "black" and "white". The names are intended to convey the expected relative intensity of the colors.

The border of the root window is initially a pixmap filled with the black-pixel. The initial background of the root window is a pixmap filled with some unspecified two-color pattern using black-pixel and white-pixel.

Min-installed-maps specifies the number of maps that can be guaranteed to be installed simultaneously (with InstallColormap), regardless of the number of entries allocated in each map. Max-installed-maps specifies the maximum number of maps that might possibly be installed simultaneously, depending on their allocations. Multiple static-visual colormaps with identical contents but differing in resource ID should be considered as a single map for the purposes of this number. For the typical case of a single hardware colormap, both values will be one.

Backing-stores indicates when the server supports backing stores for this screen, although it may be storage limited in the number of windows it can support at once. If save-unders is

True, the server can support the save-under mode in CreateWindow and ChangeWindowAttributes, although again it may be storage limited.

The current-input-events is what GetWindowAttributes would return for the all-event-masks for the root window.

Per visual-type information:

A given visual type might be listed for more than one depth or for more than one screen.

For PseudoColor, a pixel value indexes a colormap to produce independent RGB values; the RGB values can be changed dynamically. GrayScale is treated the same as PseudoColor, except which primary drives the screen is undefined, so the client should always store the same value for red, green, and blue in colormaps. For DirectColor, a pixel value is decomposed into separate RGB subfields, and each subfield separately indexes the colormap for the corresponding value. The RGB values can be changed dynamically. TrueColor is treated the same as DirectColor, except the colormap has predefined read-only RGB values, which are server-dependent, but provide (near-)linear increasing ramps in each primary. StaticColor is treated the same as PseudoColor, except the colormap has predefined read-only RGB values, which are server-dependent. StaticGray is treated the same as StaticColor, except the red, green, and blue values are equal for any single pixel value, resulting in shades of gray. StaticGray with a two-entry colormap can be thought of as "monochrome".

The red-mask, green-mask, and blue-mask are only defined for DirectColor and TrueColor. Each has one contiguous set of bits, with no intersections. Usually each mask has the same number of one bits.

The bits-per-rgb-value specifies the log base 2 of the number of distinct color intensity values (individually) of red, green, and blue. This number need not bear any relation to the number of colormap entries. Actual RGB values are always passed in the protocol within a 16-bit spectrum, with zero being minimum intensity and 65535 being the maximum intensity. On hardware that provides a linear zero-based intensity ramp, the following relationship exists:

hw-intensity = protocol-intensity / (65536 / total-hw-intensities)

Colormap entries are indexed from zero. The colormap-entries defines the number of available colormap entries in a newly created colormap. For DirectColor and TrueColor, this will usually be two to the power of the maximum number of one bits in red-mask, green-mask, and blue-mask.

10. Requests

CreateWindow

wid, parent: WINDOW
class: {InputOutput, InputOnly, CopyFromParent}
depth: CARD8
visual: VISUALID or CopyFromParent
x, y: INT16
width, height, border-width: CARD16
value-mask: BITMASK
value-list: LISTofVALUE

Errors: IDChoice, Window, Pixmap, Colormap, Cursor, Match, Value, Alloc

This request creates an unmapped window and assigns the identifier wid to it.

A class of CopyFromParent means the class is taken from the parent. A depth of zero for class InputOutput or CopyFromParent means the depth is taken from the parent. A visual of CopyFromParent means the visual type is taken from the parent. For class InputOutput, the visual type and depth must be a combination supported for the screen (else a Match error). The depth need not be the same as the parent, but the parent must not be of class InputOnly (else a Match error). For class InputOnly, the depth must be zero (else a Match error), and the visual must be one supported for the screen (else a Match error), but the parent may have any depth and class.

The server essentially acts as if InputOnly windows do not exist for the purposes of graphics requests, exposure processing, and VisibilityNotify events. An InputOnly window cannot be used as a drawable (as a source or destination for graphics requests). InputOnly and InputOutput windows act identically in other respects (properties, grabs, input control, and so on).

The window is placed on top in the stacking order with respect to siblings. The x and y coordinates are relative to the parent's origin and specify the position of the upper-left outer corner of the window (not the origin). The width and height specify the inside size, not including the border, and must be non-zero (else a Value error). The border-width for an InputOnly window must be zero (else a Match error).

The value-mask and value-list specify attributes of the window that are to be explicitly initialized. The possible values are:

Attribute	Type
background-pixmap	PIXMAP or None or ParentRelative
background-pixel	CARD32
border-pixmap	PIXMAP or CopyFromParent
border-pixel	CARD32
bit-gravity	BITGRAVITY
win-gravity	WINGRAVITY
backing-store	{NotUseful, WhenMapped, Always}
backing-planes	CARD32
backing-pixel	CARD32
save-under	BOOL
event-mask	SETofEVENT
do-not-propagate-mask	SETofDEVICEEVENT
override-redirect	BOOL
colormap	COLORMAP or CopyFromParent
cursor	CURSOR or None

The default values, when attributes are not explicitly initialized, are:

Attribute	Default
background-pixmap	None
border-pixmap	CopyFromParent
bit-gravity	Forget
win-gravity	NorthWest
backing-store	NotUseful
backing-planes	all ones
backing-pixel	zero
save-under	False
event-mask	{} (empty set)
do-not-propagate-mask	{} (empty set)
override-redirect	False
colormap	CopyFromParent
cursor	None

Attribute	Default

Only the following attributes are defined for InputOnly windows:

- win-gravity
- event-mask
- do-not-propagate-mask
- override-redirect
- cursor

It is a Match error to specify any other attributes for InputOnly windows.

If background-pixmap is given, it overrides the default background-pixmap. The background pixmap and the window must have the same root and the same depth (else a Match error). Any size pixmap can be used, although some sizes may be faster than others. If background None is specified, the window has no defined background. If background ParentRelative is specified, the parent's background is used, but the window must have the same depth as the parent (else a Matcherror). If the parent has background None, then the window will also have background None. A copy of the parent's background is not made. The parent's background is reexamined each time the window background is required. If background-pixel is given, it overrides the default background-pixmap and any background-pixmap given explicitly, and a pixmap of undefined size filled with background-pixel is used for the background. For a ParentRelative background, the background tile origin always aligns with the parent's background tile origin. Otherwise, the background tile origin is always the window origin.

When no valid contents are available for regions of a window, and the regions are either visible or the server is maintaining backing store, the server automatically tiles the regions with the window's background unless the window has a background of None. If the background is None, the previous screen contents are simply left in place if the contents come from an inferior window of the same depth, and otherwise the initial contents of the exposed regions are undefined. Exposure events are then generated for the regions, even if the background is None.

The border tile origin is always the same as the background tile origin. If border-pixmap is given, it overrides the default border-pixmap. The border pixmap and the window must have the same root and the same depth (else a Match error). Any size pixmap can be used, although some sizes may be faster than others. If CopyFromParent is given, the parent's border pixmap is copied (subsequent changes to the parent's border attribute do not affect the child), but the window must have the same depth as the parent (else a Match error). The pixmap might be copied by sharing the same pixmap object between the child and parent or by making a complete copy of the pixmap contents. If border-pixel is given, it overrides the default border-pixmap and any border-pixmap given explicitly, and a pixmap of undefined size filled with border-pixel is used for the border.

Output to a window is always clipped to the inside of the window, so that the border is never affected.

The bit-gravity defines which region of the window should be retained if the window is resized, and win-gravity defines how the window should be repositioned if the parent is resized; see ConfigureWindow.

A backing-store of WhenMapped advises the server that maintaining contents of obscured regions when the window is mapped would be beneficial. A backing-store of Always advises the server that maintaining contents even when the window is unmapped would be beneficial. In this case, the server may generate an exposure event when the window is created. A value of NotUseful advises the server that maintaining contents is unnecessary, although a server may still choose to maintain contents while the window is mapped. Note that if the server maintains contents, then the server should maintain complete contents, not just the region within the parent boundaries, even if the window is larger than its parent. While the server maintains contents, exposure events will not normally be generated, but the server may stop maintaining contents at any time.

If save-under is True, the server is advised that, when this window is mapped, saving the contents of windows it obscures would be beneficial.

When the contents of obscured regions of a window are being maintained, regions obscured by non-inferior windows are included in the destination (and source, when the window is the source) of graphics requests, but regions obscured by inferior windows are not included.

The backing-planes indicates (with one bits) which bit planes of the window hold dynamic data that must be preserved in backing-stores and during save-unders. The backing-pixel specifies what value to use in planes not covered by backing-planes. The server is free to only save the specified bit planes in the backing-store or save-under and regenerate the remaining planes with the specified pixel value. Any "extraneous" bits (beyond the specified depth of the window) in these values are simply ignored.

The event-mask defines which events the client is interested in for this window (or, for some event types, inferiors of the window). The do-not-propagate-mask defines which events should not be propagated to ancestor windows when no client has the event type selected in this window.

The override-redirect specifies whether map and configure requests on this window should override a SubstructureRedirect on the parent, typically to inform a window manager not to tamper with the window.

The colormap specifies the colormap that best reflects the "true" colors of the window. Servers capable of supporting multiple hardware colormaps may use this information, and window managers may use it for InstallColormap requests. The colormap must have the same visual type as the window (else a Match error). If CopyFromParent is specified, the parent's colormap is copied (subsequent changes to the parent's colormap attribute do not affect the child), but the window must have the same visual type as the parent (else a Match error), and the parent must not have a colormap of None (else a Match error). The colormap is copied by sharing the colormap object between the child and the parent, not by making a complete copy of the colormap contents.

If a cursor is specified, it will be used whenever the pointer is in the window. If None is specified, the parent's cursor will be used when the pointer is in the window, and any change in the parent's cursor will cause an immediate change in the displayed cursor.

This request generates a CreateNotify event.

The background and border pixmaps and the cursor may be freed immediately if no further explicit references to them are to be made.

Subsequent drawing into the background or border pixmap has an undefined effect on the window state. The server might or might not make a copy of the pixmap.

ChangeWindowAttributes

window: WINDOW
value-mask: BITMASK
value-list: LISTofVALUE

Errors: Window, Pixmap, Colormap, Cursor, Match, Value, Access

The value-mask and value-list specify which attributes are to be changed. The values and restrictions are the same as for CreateWindow.

Setting a new background, whether by background-pixmap or background-pixel, overrides any previous background. Setting a new border, whether by border-pixel or border-pixmap, overrides any previous border.

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

Changing the win-gravity does not affect the current position of the window.

Changing the backing-store of an obscured window to WhenMapped or Always or changing the backing-planes, backing-pixel, or save-under of a mapped window may have no immediate effect.

Multiple clients can select input on the same window; their event-masks are disjoint. When an event is generated, it will be reported to all interested clients. However, at most one client at a time can select for SubstructureRedirect, at most one client at a time can select for ResizeRedirect, and at most one client at a time can select for ButtonPress. An attempt to violate these restrictions results in an Access error.

There is only one do-not-propagate-mask for a window, not one per client.

Changing the colormap of a window (that is, defining a new map, not changing the contents of the existing map) generates a ColormapNotify event. Changing the colormap of a visible window might have no immediate effect on the screen; see InstallColormap.

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

The order in which attributes are verified and altered is server dependent. If an error is generated, a subset of the attributes may have been altered.

GetWindowAttributes

window: WINDOW

=>

visual: VISUALID
class: {InputOutput, InputOnly}
bit-gravity: BITGRAVITY
win-gravity: WINGRAVITY
backing-store: {NotUseful, WhenMapped, Always}
backing-planes: CARD32
backing-pixel: CARD32
save-under: BOOL
colormap: COLORMAP or None
map-is-installed: BOOL
map-state: {Unmapped, Unviewable, Viewable}
all-event-masks, your-event-mask: SETofEVENT
do-not-propagate-mask: SETofDEVICEEVENT
override-redirect: BOOL

Errors: Window

This request returns current attributes of the window. A window is Unviewable if it is mapped but some ancestor is unmapped. All-event-masks is the inclusive-OR of all event masks selected on the window by clients. Your-event-mask is the event mask selected by the querying client.

DestroyWindow

window: WINDOW

Errors: Window

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

Normal exposure processing on formerly obscured windows is performed.

If the window is a root window, this request has no effect.

DestroySubwindows

window: WINDOW

Errors: Window

This request performs a DestroyWindow request on all children of the window, in bottom to top stacking order.

ChangeSaveSet

window: WINDOW
mode: {Insert, Delete}

Errors: Window, Match, Value

This request adds or removes the specified window from the client's "save-set". The window must have been created by some other client (else a Match error). For further information about the use of the save-set, see Section 11.

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

ReparentWindow

window, *parent*: WINDOW
x, *y*: INT16

Errors: Window, Match

If the window is mapped, an UnmapWindow request is performed automatically first. The window is then removed from its current position in the hierarchy and is inserted as a child of the specified parent. The x and y coordinates are relative to the parent's origin and specify the new position of the upper-left outer corner of the window. The window is placed on top in the stacking order with respect to siblings. A ReparentNotify event is then generated. The override-redirect attribute of the window is passed on in this event; a value of True indicates that a window manager should not tamper with this window. Finally, if the window was originally mapped, a MapWindow request is performed automatically.

Normal exposure processing on formerly obscured windows is performed. The server might not generate exposure events for regions from the initial unmap that are immediately obscured by the final map.

A Match error is generated if the:

- New parent is not on the same screen as the old parent
- New parent is the window itself or an inferior of the window
- Window has a ParentRelative background, and the new parent is not the same depth as the window.

MapWindow

window: WINDOW

Errors: Window

If the window is already mapped, this request has no effect.

If the override-redirect attribute of the window is False and some other client has selected SubstructureRedirect on the parent, then a MapRequest event is generated, but the window remains unmapped. Otherwise, the window is mapped and a MapNotify event is generated.

If the window is now viewable and its contents had been discarded, the window is tiled with its background (if no background is defined, the existing screen contents are not altered) and one or more exposure events are generated. If a backing-store has been maintained while the window was unmapped, no exposure events are generated. If a backing-store will now be maintained, a full-window exposure is always generated. Otherwise, only visible regions may be reported. Similar tiling and exposure take place for any newly viewable inferiors.

MapSubwindows

window: WINDOW

Errors: Window

This request performs a MapWindow request on all unmapped children of the window, in top to bottom stacking order.

UnmapWindow

window: WINDOW

Errors: Window

If the window is already unmapped, this request has no effect. Otherwise, the window is unmapped, and an UnmapNotify event is generated. Normal exposure processing on formerly obscured windows is performed.

UnmapSubwindows

window: WINDOW

Errors: Window

This request performs an UnmapWindow request on all mapped children of the window, in bottom to top stacking order.

ConfigureWindow

window: WINDOW
value-mask: BITMASK
value-list: LISTofVALUE

Errors: Window, Match, Value

This request changes the configuration of the window. The value-mask and value-list specify which values are to be given. The possible values are:

Attribute	Type
x	INT16
y	INT16
width	CARD16
height	CARD16
border-width	CARD16
sibling	WINDOW
stack-mode	{Above, Below, TopIf, BottomIf, Opposite}

The x and y coordinates are relative to the parent's origin and specify the position of the upper-left outer corner of the window. The width and height specify the inside size, not including the border, and must be non-zero (else a Value error). Values not specified are taken from the existing geometry of the window. Note that changing just the border-width leaves the outer left corner of the window in a fixed position but moves the absolute position of the window's origin. It is a Match error to attempt to make the border-width of an InputOnly window non-zero.

If the override-redirect attribute of the window is False and some other client has selected SubstructureRedirect on the parent, a ConfigureRequest event is generated, and no further processing is performed. Otherwise, the following is performed.

If some other client has selected ResizeRedirect on the window and the inside width or height of the window is being changed, a ResizeRequest event is generated, and the current inside width and height are used instead in the following. Note that the override-redirect attribute of the window has no effect on ResizeRedirect, and that SubstructureRedirect on the parent has precedence over ResizeRedirect on the window.

The geometry of the window is changed as specified, and the window is restacked among siblings as described below, and a ConfigureNotify event is generated if the state of the window actually changes. If the inside width or height of the window has actually changed, then children of the window are affected as described below. Exposure processing is performed on formerly obscured windows (including the window itself and its inferiors, if regions of them were obscured but now are not). Exposure processing is also performed on any new regions of the window (as a result of increasing the width or height) and any regions where window contents are lost.

If the inside width or height of a window is not changed, but the window is moved (or its border is changed), then the contents of the window are not lost but move with the window. Changing the inside width or height of the window causes its contents to be moved or lost, depending on the bit-gravity of the window, and causes children to be reconfigured, depending on their win-gravity. For a change of width and height of W and H, we define the [x, y] pairs:

Direction	Deltas
NorthWest	[0, 0]
North	[W/2, 0]
NorthEast	[W, 0]
West	[0, H/2]
Center	[W/2, H/2]
East	[W, H/2]
SouthWest	[0, H]
South	[W/2, H]
SouthEast	[W, H]

When a window with one of these bit-gravities is resized, the corresponding pair defines the change in position of each pixel in the window. When a window with one of these win-gravities has its parent window resized, the corresponding pair defines the change in position of the window within the parent. When a window is so repositioned, a GravityNotifyevent is generated. GravityNotify events are generated after the ConfigureNotify event is generated.

A gravity of Static indicates that the contents or origin should not move relative to the origin of the root window. If the change in size of the window is coupled with a change in position of [X, Y], then for bit-gravity the change in position of each pixel is [–X, –Y], and for win-gravity the change in position of a child when its parent is so resized is [–X, –Y]. Note that Static gravity still only takes effect when the width or height of the window is changed, not when the window is simply moved.

A bit-gravity of Forget indicates that the window contents are always discarded after a size change (even if backing-store or save-under has been requested). The window is tiled with its background (if no background is defined, the existing screen contents are not altered) and one or more exposure events are generated. A server may also ignore the specified bit-gravity and use Forget instead.

A win-gravity of Unmap is like NorthWest, but the child is also unmapped when the parent is resized, and an UnmapNotify event is generated. UnmapNotify events are generated after the ConfigureNotify event is generated.

If a sibling and a stack-mode is specified, the window is restacked as follows:

Above	Window is placed just above sibling.
Below	Window is placed just below sibling.
TopIf	If sibling occludes window, then window is placed at the top of the stack.
BottomIf	If window occludes sibling, then window is placed at the bottom of the stack.
Opposite	If sibling occludes window, then window is placed at the top of the stack. Otherwise, if window occludes sibling, then window is placed at the bottom of the stack.

If a stack-mode is specified but no sibling is specified, the window is restacked as follows:

Above	Window is placed at the top of the stack.
Below	Window is placed at the bottom of the stack.
TopIf	If any sibling occludes window, then window is placed at the top of the stack.
BottomIf	If window occludes any sibling, then window is placed at the bottom of the stack.
Opposite	If any sibling occludes window, then window is placed at the top of the stack. Otherwise, if window occludes any sibling, then window is placed at the bottom of the stack.

It is a Match error if a sibling is specified without a stack-mode or if the window is not actually a sibling.

Note that the computations for BottomIf, TopIf, and Opposite are performed with respect to the window's final geometry (as controlled by the other arguments to the request), not its initial geometry.

Attempts to configure a root window have no effect.

CirculateWindow

window: WINDOW
direction: {RaiseLowest, LowerHighest}

Errors: Window, Value

If some other client has selected SubstructureRedirect on the window, then a CirculateRequest event is generated, and no further processing is performed. Otherwise, the following is performed, and then a CirculateNotify event is generated if the window is actually restacked.

For RaiseLowest, CirculateWindow raises the lowest mapped child (if any) that is occluded by another child to the top of the stack. For LowerHighest, CirculateWindow 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.

GetGeometry

drawable: DRAWABLE

=>

root: WINDOW
depth: CARD8

x, y: INT16
width, height, border-width: CARD16

Errors: Drawable

This request returns the root and (current) geometry of the drawable. The depth is the number of bits per pixel for the object. The x, y, and border-width will always be zero for pixmaps. For a window, the x and y coordinates specify the upper left outer corner of the window relative to its parent's origin, and the width and height specify the inside size (not including the border).

It is legal to pass an InputOnly window as a drawable to this request.

QueryTree

window: WINDOW

=>

root: WINDOW
parent: WINDOW or None
children: LISTofWINDOW

Errors: Window

This request returns the root, the parent, and children of the window. The children are listed in bottom-to-top stacking order.

InternAtom

name: STRING8
only-if-exists: BOOL

=>

atom: ATOM or None

Errors: Value, Alloc

This request returns the atom for the given name. If only-if-exists is False, then the atom is created if it does not exist. The string should use the ISO Latin-1 encoding, and upper/lower case matters.

The lifetime of an atom is not tied to the interning client. Atoms remained defined until server reset (see Section 11).

GetAtomName

atom: ATOM

=>

name: STRING8

Errors: Atom

This request returns the name for the given atom.

ChangeProperty

window: WINDOW
property, *type*: ATOM
format: {8, 16, 32}
mode: {Replace, Prepend, Append}
data: LISTofINT8 or LISTofINT16 or LISTofINT32

Errors: Window, Atom, Value, Match, Alloc

This request alters the property for the specified window. The type is uninterpreted by the server. The format specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities, so that the server can correctly byte-swap as necessary.

If mode is Replace, the previous property value is discarded. If the mode is Prepend or Append, then the type and format must match the existing property value (else a Match error). If the property is undefined, it is treated as defined with the correct type and format with zero-length data. For Prepend, the data is tacked on to the beginning of the existing data, and, for Append, it is tacked on to the end of the existing data.

This request generates a PropertyNotify event on the window.

The lifetime of a property is not tied to the storing client. Properties remain until explicitly deleted, or the window is destroyed, or until server reset (see Section 11).

The maximum size of a property is server dependent and may vary dynamically.

DeleteProperty

window: WINDOW
property: ATOM

Errors: Window, Atom

This request deletes the property from the specified window if the property exists and generates a PropertyNotify event on the window unless the property does not exist.

GetProperty

window: WINDOW
property: ATOM
type: ATOM or AnyPropertyType
long-offset, long-length: CARD32
delete: BOOL

=>

type: ATOM or None
format: {0, 8, 16, 32}
bytes-after: CARD32
value: LISTofINT8 or LISTofINT16 or LISTofINT32

Errors: Window, Atom, Value

If the specified property does not exist for the specified window, then the return type is None, the format and bytes-after are zero, and the value is empty. The delete argument is ignored in this case. If the specified property exists but its type does not match the specified type, then the return type is the actual type of the property, the format is the actual format of the property (never zero), the bytes-after is the length of the property in bytes (even if the format is 16 or 32), and the value is empty. The delete argument is ignored in this case. If the specified property exists, and either AnyPropertyType is specified or the specified type matches the actual type of the property, then the return type is the actual type of the property, the format is the actual format of the property (never zero), and the bytes-after and value are as follows. Define the following:

N = actual length of the stored property in bytes
(even if the format is 16 or 32)
I = 4 * long-offset
T = N – I
L = MINIMUM(T, 4 * long-length)
A = N – (I + L)

The returned value starts at byte index I in the property (indexing from 0), and its length in bytes is L. However, it is a Value error if long-offset is given such that L is negative. The value of bytes-after is A, giving the number of trailing unread bytes in the stored property. If delete is True and the bytes-after is zero, the property is also deleted from the window, and a PropertyNotify event is generated on the window.

RotateProperties

window: WINDOW
delta: INT16
properties: LISTofATOM

Errors: Window, Atom, Match

If the property names in the list are viewed as being numbered starting from zero, and there are N property names in the list, then the value associated with property name I becomes the value associated with property name (I + delta) mod N, for all I from zero to N – 1. The effect is to rotate the states by delta places around the virtual ring of property names (right for positive delta, left for negative delta).

If delta mod N is non-zero, a PropertyNotify event is generated for each property, in the order listed.

If an atom occurs more than once in the list or no property with that name is defined for the window, a Match error is generated. If an Atom or Match error is generated, no properties are changed.

ListProperties

window: WINDOW

=>

atoms: LISTofATOM

Errors: Window

This request returns the atoms of properties currently defined on the window.

SetSelectionOwner

selection: ATOM
owner: WINDOW or None
time: TIMESTAMP or CurrentTime

Error: Atom, Window

This request changes the owner, owner window, and last-change time of the specified selection. This request has no effect if the specified time is earlier than the current last-change time of the specified selection or is later than the current server time. Otherwise, the last-change time is set to the specified time, with CurrentTime replaced by the current server time. If the owner window is specified as None, then the owner of the selection becomes None (that is, no owner). Otherwise, the owner of the selection becomes the client executing the request. If the new owner (whether a client or None) is not the same as the current owner, and the current owner is not None, then the current owner is sent a SelectionClear event.

If the client that is the owner of a selection is later terminated (that is, its connection is closed) or if the owner window it has specified in the request is later destroyed, then owner of the selection automatically reverts to None, but the last-change time is not affected.

The selection atom is uninterpreted by the server. The owner window is returned by the GetSelectionOwner request and is reported in SelectionRequest and SelectionClear events.

Selections are global to the server.

GetSelectionOwner

selection: ATOM

=>

owner: WINDOW or None

Errors: Atom

This request returns the current owner window of the specified selection, if any. If None is returned, then there is no owner for the selection.

ConvertSelection

selection, *target*: ATOM
property: ATOM or None
requestor: WINDOW
time: TIMESTAMP or CurrentTime

Error: Atom, Window

If the specified selection has an owner, the server sends a SelectionRequest event to that owner. If no owner for the specified selection exists, the server generates a SelectionNotify event to the requestor with property None. The arguments are passed on unchanged in either event.

SendEvent

destination: WINDOW or PointerWindow or InputFocus
propagate: BOOL
event-mask: SETofEVENT
event: <normal-event-format>

Errors: Window, Value

If PointerWindow is specified, destination is replaced with the window that the pointer is in. If InputFocus is specified, then if the focus window contains the pointer, destination is replaced with the window that the pointer is in. Otherwise, destination is replaced with the focus window.

If the event-mask is the empty set, then the event is sent to the client that created the destination window. If that client no longer exists, no event is sent.

If propagate is False, then the event is sent to every client selecting on destination any of the event types in event-mask.

If propagate is True and no clients have selected on destination any of the event types in event-mask, then destination is replaced with the closest ancestor of destination for which some client has selected a type in event-mask and no intervening window has that type in its do-not-propagate-mask. If no such window exists, or if the window is an ancestor of the focus window and InputFocus was originally specified as the destination, then the event is not sent to any clients. Otherwise, the event is reported to every client selecting on the final destination any of the types specified in event-mask.

The event code must be one of the core events or one of the events defined by an extension, so that the server can correctly byte swap the contents as necessary. The contents of the event are otherwise unaltered and unchecked by the server except to force on the most significant bit of the event code and to set the sequence number in the event correctly.

Active grabs are ignored for this request.

GrabPointer

grab-window: WINDOW
owner-events: BOOL
event-mask: SETofPOINTEREVENT
pointer-mode, *keyboard-mode*: {Synchronous, Asynchronous}
confine-to: WINDOW or None
cursor: CURSOR or None
time: TIMESTAMP or CurrentTime

=>

status: {Success, AlreadyGrabbed, Frozen, InvalidTime, NotViewable}

Errors: Cursor, Window, Value

This request actively grabs control of the pointer. Further pointer events are only reported to the grabbing client. The request overrides any active pointer grab by this client.

If owner-events is False, all generated pointer events are reported with respect to grab-window and are 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 simply discarded.

If pointer-mode is Asynchronous, pointer event processing continues normally. If the pointer is currently frozen by this client, then processing of pointer events is resumed. If pointer-mode is Synchronous, the state of the pointer (as seen by means of the protocol) appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing AllowEvents request or until the pointer grab is released. Actual pointer changes are not lost while the pointer is frozen. They are simply queued for later processing.

If keyboard-mode is Asynchronous, keyboard event processing is unaffected by activation of the grab. If keyboard-mode is Synchronous, the state of the keyboard (as seen by means of the protocol) appears to freeze, and no further keyboard events are generated by the server until the grabbing client issues a releasing AllowEvents request or until the pointer grab is released. Actual keyboard changes are not lost while the keyboard is frozen. They are simply queued for later processing.

If a cursor is specified, then it is displayed regardless of what window the pointer is in. If no cursor is specified, then when the pointer is in grab-window or one of its subwindows, the normal cursor for that window is displayed, and otherwise the cursor for grab-window is displayed.

If a confine-to window is specified, then the pointer will be restricted to stay contained in that window. The confine-to window need have no relationship to the grab-window. If the pointer is not initially in the confine-to window, then it is warped automatically to the closest edge (and enter/leave events generated normally) just before the grab activates. If the confine-to window is subsequently reconfigured, the pointer will be warped automatically as necessary to keep it contained in the window.

This request generates EnterNotify and LeaveNotify events.

The request fails with status AlreadyGrabbed if the pointer is actively grabbed by some other client. The request fails with status Frozen if the pointer is frozen by an active grab of another client. The request fails with status NotViewable if grab-window or confine-to window is not viewable. The request fails with status InvalidTime if the specified time is earlier than the last-pointer-grab time or later than the current server time. Otherwise the last-pointer-grab time is set to the specified time, with CurrentTime replaced by the current server time.

UngrabPointer

time: TIMESTAMP or CurrentTime

This request releases the pointer if this client has it actively grabbed (from either GrabPointer or GrabButton or from a normal button press) and releases any queued events. The request has no effect if the specified time is earlier than the last-pointer-grab time or is later than the current server time.

This request generates EnterNotify and LeaveNotify events.

An UngrabPointer request is performed automatically if the event window or confine-to window for an active pointer grab becomes not viewable.

GrabButton

modifiers: SETofKEYMASK or AnyModifier
button: BUTTON or AnyButton
grab-window: WINDOW
owner-events: BOOL
event-mask: SETofPOINTEREVENT
pointer-mode, keyboard-mode: {Synchronous, Asynchronous}
confine-to: WINDOW or None
cursor: CURSOR or None

Errors: Cursor, Window, Value, Access

This request establishes a passive grab. In the future, if the pointer is not grabbed and the specified button is logically pressed when the specified modifier keys are logically down, and no other buttons or modifier keys are logically down, and grab-window contains the pointer, and the confine-to window (if any) is viewable, and a passive grab on the same button/key combination does not exist on any ancestor of grab-window, then the pointer is actively grabbed as described in GrabPointer, the last-pointer-grab time is set to the time at which the button was pressed (as transmitted in the ButtonPress event), and the ButtonPress event is reported. The interpretation of the remaining arguments is as for GrabPointer. The active grab is terminated automatically when he logical state of the pointer has all buttons released, independent of the logical state of modifier keys. Note that the logical state of a device (as seen by means of the protocol) may lag the physical state, if device event processing is frozen.

A modifier of AnyModifier is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned keycodes. A button of AnyButton is equivalent to issuing the request for all possible buttons. Otherwise, it is not required that the button specified currently be assigned to a physical button.

An Access error is generated if some other client has already issued a GrabButton request with the same button/key combination on the same window. When using AnyModifier or AnyButton, the request fails completely (no grabs are established), and an Access error is generated if there is a conflicting grab for any combination. The request has no effect on an active grab.

UngrabButton

modifiers: SETofKEYMASK or AnyModifier
button: BUTTON or AnyButton
grab-window: WINDOW

Errors: Window

This request releases the passive button/key combination on the specified window if it was grabbed by this client. A modifiers of AnyModifier is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). A button of AnyButton is equivalent to issuing the request for all possible buttons. The request has no effect on an active grab.

ChangeActivePointerGrab

event-mask: SETofPOINTEREVENT
cursor: CURSOR or None
time: TIMESTAMP or CurrentTime

Errors: Cursor

This request changes the specified dynamic parameters if the pointer is actively grabbed by the client and the specified time is no earlier than the last-pointer-grab time and no later than the current server time. The interpretation of event-mask and cursor are as in GrabPointer. This request has no effect on the passive parameters of a GrabButton.

GrabKeyboard

grab-window: WINDOW
owner-events: BOOL
pointer-mode, keyboard-mode: {Synchronous, Asynchronous}
time: TIMESTAMP or CurrentTime

=>

status: {Success, AlreadyGrabbed, Frozen, InvalidTime, NotViewable}

Errors: Window, Value

This request actively grabs control of the keyboard. Further key events are reported only to the grabbing client. This request overrides any active keyboard grab by this client.

If owner-events is False, all generated key events are reported with respect to grab-window. If owner-events is True, then if a generated key event would normally be reported to this client, it is reported normally. Otherwise, the event is reported with respect to the grab-window. Both KeyPress and KeyRelease events are always reported, independent of any event selection made by the client.

If keyboard-mode is Asynchronous, keyboard event processing continues normally. If the keyboard is currently frozen by this client, then processing of keyboard events is resumed. If keyboard-mode is Synchronous, the state of the keyboard (as seen by means of the protocol) appears to freeze, and no further keyboard events are generated by the server until the grabbing client issues a releasing AllowEvents request or until the keyboard grab is released. Actual keyboard changes are not lost while the keyboard is frozen. They are simply queued for later processing.

If pointer-mode is Asynchronous, pointer event processing is unaffected by activation of the grab. If pointer-mode is Synchronous, the state of the pointer (as seen by means of the protocol) appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing AllowEvents request or until the keyboard grab is released. Actual pointer changes are not lost while the pointer is frozen. They are simply queued for later processing.

This request generates FocusIn and FocusOut events.

The request fails with status AlreadyGrabbed if the keyboard is actively grabbed by some other client. The request fails with status Frozen if the keyboard is frozen by an active grab of another client. The request fails with status NotViewable if grab-window is not viewable. The request fails with status InvalidTime if the specified time is earlier than the last-keyboard-grab time or later than the current server time. Otherwise, the last-keyboard-grab time is set to the specified time, with CurrentTime replaced by the current server time.

UngrabKeyboard

time: TIMESTAMP or CurrentTime

This request releases the keyboard if this client has it actively grabbed (from either GrabKeyboard or GrabKey) and releases any queued events. The request has no effect if the specified time is earlier than the last-keyboard-grab time or is later than the current server time.

This request generates FocusIn and FocusOut events.

An UngrabKeyboard is performed automatically if the event window for an active keyboard grab becomes not viewable.

GrabKey

key: KEYCODE or AnyKey
modifiers: SETofKEYMASK or AnyModifier
grab-window: WINDOW
owner-events: BOOL

pointer-mode, *keyboard-mode*: {Synchronous, Asynchronous}

Errors: Window, Value, Access

This request establishes a passive grab on the keyboard. In the future, if the keyboard is not grabbed and the specified key (which can itself be a modifier key) is logically pressed when the specified modifier keys are logically down, and no other modifier keys are logically down, and either a) the grab-window is an ancestor of (or is) the focus window, or b) 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 grab-window, then the keyboard is actively grabbed as described in GrabKeyboard, 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 interpretation of the remaining arguments is as for GrabKeyboard. The active grab is terminated automatically when the logical state of the keyboard has the specified key released, independent of the logical state of modifier keys. Note that the logical state of a device (as seen by means of the protocol) may lag the physical state, if device event processing is frozen.

A modifier of AnyModifier is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned keycodes. A key of AnyKey is equivalent to issuing the request for all possible keycodes. Otherwise, the key must be in the range specified by min-keycode and max-keycode in the connection setup (else a Value error).

An Access error is generated if some other client has issued a GrabKey with the same key combination on the same window. When using AnyModifier or AnyKey, the request fails completely (no grabs are established), and an Access error is generated if there is a conflicting grab for any combination.

UngrabKey

key: KEYCODE or AnyKey
modifiers: SETofKEYMASK or AnyModifier
grab-window: WINDOW

Errors: Window

This request releases the key combination on the specified window if it was grabbed by this client. A modifiers of AnyModifier is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). A key of AnyKey is equivalent to issuing the request for all possible keycodes. This request has no effect on an active grab.

AllowEvents

mode: {AsyncPointer, SyncPointer, ReplayPointer, AsyncKeyboard, SyncKeyboard,
ReplayKeyboard, AsyncBoth, SyncBoth}
time: TIMESTAMP or CurrentTime

Errors: Value

This request releases some queued events if the client has caused a device to freeze. The request has no effect if the specified time is earlier than the last-grab time of the most recent active grab for the client or if the specified time is later than the current server time.

For AsyncPointer, if the pointer is frozen by the client, pointer event processing continues normally. If the pointer is frozen twice by the client on behalf of two separate grabs, AsyncPointer "thaws" for both. AsyncPointer has no effect if the pointer is not frozen by the client, but the pointer need not be grabbed by the client.

For SyncPointer, if the pointer is frozen and actively grabbed by the client, pointer event processing continues normally until the next ButtonPress or ButtonRelease event is reported to the client, at which 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. SyncPointer has no effect if the pointer is not frozen by the client or if the pointer is not grabbed by the client.

For ReplayPointer, if the pointer is actively grabbed by the client and is frozen as the result of an event having been sent to the client (either from the activation of a GrabButton or AllowEvents with mode SyncPointer, but not from a GrabPointer), then the pointer grab is released and that event is completely reprocessed, but this time ignoring any passive grabs at or above (towards the root) the grab-window of the grab just released. The request has no effect if the pointer is not grabbed by the client or if the pointer is not frozen as the result of an event.

For AsyncKeyboard, if the keyboard is frozen by the client, keyboard event processing continues normally. If the keyboard is frozen twice by the client on behalf of two separate grabs, AsyncKeyboard "thaws" for both. AsyncKeyboard has no effect if the keyboard is not frozen by the client, but the keyboard need not be grabbed by the client.

For SyncKeyboard, if the keyboard is frozen and actively grabbed by the client, keyboard event processing continues normally until the next KeyPress or KeyRelease event is reported to the client, at which 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. SyncKeyboard has no effect if the keyboard is not frozen by the client or if the keyboard is not grabbed by the client.

For ReplayKeyboard, if the keyboard is actively grabbed by the client and is frozen as the result of an event having been sent to the client (either from the activation of a GrabKey or from a previous AllowEvents with mode SyncKeyboard, but not from a GrabKeyboard), then the keyboard grab is released and that event is completely reprocessed, but this time ignoring any passive grabs at or above (towards the root) the grab-window of the grab just released. The request has no effect if the keyboard is not grabbed by the client or if the keyboard is not frozen as the result of an event.

For SyncBoth, if both pointer and keyboard are frozen by the client, event processing (for both devices) continues normally until the next ButtonPress, ButtonRelease, KeyPress, or KeyRelease event is reported to the client for a grabbed device (button event for the pointer, key event for the keyboard), at which time the devices again appear to freeze. However, if the reported event causes the grab to be released, then the devices do not freeze (but if the other device is still grabbed, then a subsequent event for it will still cause both devices to freeze). SyncBoth has no effect unless both pointer and keyboard are frozen by the client. If the pointer of 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).

For AsyncBoth, if the pointer and the keyboard are frozen by the client, event processing (for both devices) continues normally. If a device is frozen twice by the client on behalf of two separate grabs, AsyncBoth "thaws" for both. AsyncBoth has no effect unless both pointer and keyboard are frozen by the client.

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

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

GrabServer

This request disables processing of requests and close-downs on all other connections (than the one this request arrived on).

UngrabServer

This request restarts processing of requests and close-downs on other connections.

QueryPointer

window: WINDOW

=>

root: WINDOW
child: WINDOW or None
same-screen: BOOL
root-x, root-y, win-x, win-y: INT16
mask: SETofKEYBUTMASK

Errors: Window

The root window the pointer is logically on and pointer coordinates relative to the root's origin are returned. If same-screen is False, then the pointer is not on the same screen as the argument window, and child is None and win-x and win-y are zero. If same-screen is True, then win-x and win-y are the pointer coordinates relative to the argument window's origin, and child is the child containing the pointer, if any. The current logical state of the modifier keys and the buttons are also returned. Note that the logical state of a device (as seen by means of the protocol) may lag the physical state, if device event processing is frozen.

GetMotionEvents

start, stop: TIMESTAMP or CurrentTime
window: WINDOW

=>

events: LISTofTIMECOORD

where

TIMECOORD:	{x, y: CARD16 time: TIMESTAMP}

Error: Window

This request 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 are reported relative to the origin of the window.

If the start time is later than the stop time or if the start time is in the future, no events are returned. If the stop time is in the future, it is equivalent to specifying CurrentTime.

TranslateCoordinates

src-window, dst-window: WINDOW
src-x, src-y: INT16

=>

same-screen: BOOL
child: WINDOW or None
dst-x, dst-y: INT16

Errors: Window

The src-x and src-y coordinates are taken relative to src-window's origin and are returned as dst-x and dst-y coordinates relative to dst-window's origin. If same-screen is False, then src-window and dst-window are on different screens, and dst-x and dst-y are zero. If the coordinates are contained in a mapped child of dst-window, then that child is returned.

WarpPointer

src-window: WINDOW or None
dst-window: WINDOW or None
src-x, *src-y*: INT16
src-width, *src-height*: CARD16
dst-x, *dst-y*: INT16

Errors: Window

If dst-window is None, this request moves the pointer by offsets [dst-x, dst-y] relative to the current position of the pointer. If dst-window is a window, this request moves the pointer to [dst-x, dst-y] relative to dst-window's origin. However, if src-window is not None, the move only takes place if if src-window contains the pointer and the pointer is contained in the specified rectangle of src-window.

The src-x and src-y coordinates are relative to src-window's origin. If src-height is zero, it is replaced with the current height of src-window minus src-y. If src-width is zero, it is replaced with the current width of src-window minus src-x.

This request cannot be used to move the pointer outside the confine-to window of an active pointer grab. An attempt will only move the pointer as far as the closest edge of the confine-to window.

This request will generate events just as if the user had (instantaneously) moved the pointer.

SetInputFocus

focus: WINDOW or PointerRoot or None
revert-to: {Parent, PointerRoot, None}
time: TIMESTAMP or CurrentTime

Errors: Window, Value, Match

This request changes the input focus and the last-focus-change time. The request has no effect if the specified time is earlier than the current last-focus-change time or is later than the current server time. Otherwise, the last-focus-change time is set to the specified time, with CurrentTime replaced by the current server time.

If None is specified as the focus, all keyboard events are discarded until a new focus window is set. In this case, the revert-to argument is ignored.

If a window is specified as the focus, it becomes the keyboard's focus window. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported normally. Otherwise, the event is reported with respect to the focus window.

If PointerRoot is specified as the focus, the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event. In this case, the revert-to argument is ignored.

This request generates FocusIn and FocusOut events.

The specified focus window must be viewable at the time of the request (else a Match error). If the focus window later becomes not viewable, the new focus window depends on the revert-to argument. If revert-to is Parent, the focus reverts to the parent (or the closest viewable ancestor) and the new revert-to value is take to be None. If revert-to is PointerRoot or None, the focus reverts to that value. When the focus reverts, FocusIn and FocusOut events are generated, but the last-focus-change time is not affected.

GetInputFocus

=>

focus: WINDOW or PointerRoot or None
revert-to: {Parent, PointerRoot, None}

This request returns the current focus state.

QueryKeymap

=>

keys: LISTofCARD8

This request returns a bit vector for the logical state of the keyboard. Each one bit indicates that the corresponding key is currently pressed. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N+7, with the least significant bit in the byte representing key 8N. Note that the logical state of a device (as seen by means of the protocol) may lag the physical state, if device event processing is frozen.

OpenFont

fid: FONT
name: STRING8

Errors: IDChoice, Name, Alloc

This request loads the specified font, if necessary, and associates identifier fid with it. The font name should use the ISO Latin-1 encoding, and upper/lower case does not matter.

Fonts are not associated with a particular screen and can be stored as a component of any graphics context.

CloseFont

font: FONT

Errors: Font

This request deletes the association between the resource ID and the font. The font itself will be freed when no other resource references it.

QueryFont

font: FONTABLE

=>

font-info: FONTINFO
char-infos: LISTofCHARINFO

where

```
FONTINFO:     [draw-direction: { LeftToRight, RightToLeft}
               min-char-or-byte2, max-char-or-byte2: CARD16
               min-byte1, max-byte1: CARD8
               all-chars-exist: BOOL
               default-char: CARD16
               min-bounds: CHARINFO
               max-bounds: CHARINFO
               font-ascent: INT16
               font-descent: INT16
               properties: LISTofFONTPROP]
FONTPROP:     [name: ATOM
               value: <32-bit-value>]
CHARINFO:     [left-side-bearing: INT16
               right-side-bearing: INT16
               character-width: INT16
               ascent: INT16
               descent: INT16
               attributes: CARD16]
```

Errors: Font

This request returns logical information about a font. If a gcontext is given, the currently contained font is used.

The draw-direction is just a hint and indicates whether most char-infos have a positive, LeftToRight, or a negative, RightToLeft, character-width metric. The core protocol defines no support for vertical text.

If min-byte1 and max-byte1 are both zero, then min-char-or-byte2 specifies the linear character index corresponding to the first element of char-infos, and max-char-or-byte2 specifies the linear character index of the last element. If either min-byte1 or max-byte1 are non-zero, then both min-char-or-byte2 and max-char-or-byte2 will be less than 256, and the 2-byte character index values corresponding to char-infos element N (counting from 0) are:

> byte1 N/D + min-byte1
> byte2 = N\D + min-char-or-byte2

where

> D = max-char-or-byte2 – min-char-or-byte2 + 1
> / = integer division
> \ = integer modulus

If char-infos has length zero, then min-bounds and max-bounds will be identical, and the effective char-infos is one filled with this char-info, of length:

> L = D * (max-byte1 – min-byte1 + 1)

That is, all glyphs in the specified linear or matrix range have the same information, as given by min-bounds (and max-bounds). If all-chars-exist is True, then all characters in char-infos have non-zero bounding boxes.

The default-char specifies the character that will be used when an undefined or non-existent character is used. Note that default-char is a CARD16, not CHAR2B. For a font using 2-byte matrix format, the default-char has byte1 in the most significant byte and byte2 in the least significant byte. If the default-char itself specifies an undefined or non-existent character, then no printing is performed for an undefined or non-existent character.

The min-bounds and max-bounds contain the minimum and maximum values of each individual CHARINFO component over all char-infos (ignoring non-existent characters). The bounding box of the font (that is, the smallest rectangle enclosing the shape obtained by superimposing all characters at the same origin [x,y]) has its upper-left coordinate at

> [x + min-bounds.left-side-bearing, y – max-bounds.ascent]

with a width of

> max-bounds.right-side-bearing – min-bounds.left-side-bearing

and a height of

> max-bounds.ascent + max-bounds.descent

The font-ascent is the logical extent of the font above the baseline, for determining line spacing. Specific characters may extend beyond this. The font-descent is the logical extent of the font at or below the baseline for determining line spacing. Specific characters may extend beyond this. If the baseline is at Y-coordinate y, then the logical extent of the font is inclusive between the Y-coordinate values (y – font-ascent) and (y + font-descent – 1).

A font is not guaranteed to have any properties. The interpretation of the property value (for example, INT32, CARD32) must be derived from a priori knowledge of the property. When possible, fonts should have at least the following properties (note that the trailing colon is not part of the name, and upper/lower case matters).

Property	Type	Description
MIN_SPACE	CARD32	The minimum interword spacing, in pixels.
NORM_SPACE	CARD32	The normal interword spacing, in pixels.
MAX_SPACE	CARD32	The maximum interword spacing, in pixels.
END_SPACE	CARD32	The additional spacing at the end of sentences, in pixels.
SUPERSCRIPT_X SUPERSCRIPT_Y	INT32	Offsets from the character origin where superscripts should begin, in pixels. If the origin is at [x,y], then superscripts should begin at [x + SUPERSCRIPT_X, y – SUPERSCRIPT_Y].
SUBSCRIPT_X SUBSCRIPT_Y	INT32	Offsets from the character origin where subscripts should begin, in pixels. If the origin is at [x,y], then subscripts should begin at [x + SUBSCRIPT_X, y + SUBSCRIPT_Y].
UNDERLINE_POSITION	INT32	Y offset from the baseline to the top of an underline, in pixels. If the baseline is Y-coordinate y, then the top of the underline is at (y + UNDERLINE_POSITION).
UNDERLINE_THICKNESS	CARD32	Thickness of the underline, in pixels.
STRIKEOUT_ASCENT STRIKEOUT_DESCENT	INT32	Vertical extents for boxing or voiding characters, in pixels. If the baseline is at Y-coordinate y, then the top of the strikeout box is at (y – STRIKEOUT_ASCENT), and the height of the box is (STRIKEOUT_ASCENT + STRIKEOUT_DESCENT).
ITALIC_ANGLE	INT32	The angle of the dominant staffs of characters in the font, in degrees scaled by 64, relative to the three-oclock position from the character origin, with positive indicating counterclockwise motion (as in Arc requests).
X_HEIGHT	INT32	"1 ex" as in TeX, but expressed in units of pixels. Often the height of lowercase x.
QUAD_WIDTH	INT32	"1 em" as in TeX, but expressed in units of pixels. Often the width of the digits 0-9.
CAP_HEIGHT	INT32	Y offset from the baseline to the top of the capital letters, ignoring accents, in pixels. If the baseline is at Y-coordinate y, then the top of the capitals is at (y - CAP_HEIGHT).
WEIGHT	CARD32	The weight or boldness of the font, expressed as a value between 0 and 1000.
POINT_SIZE	CARD32	The point size, expressed in 1/10ths, of this font at the ideal resolution.

Property	Type	Description
RESOLUTION	CARD32	The number of pixels per point, expressed in 1/100ths, at which this font was created.

For a character origin at [x,y], the bounding box of a character, that is, the smallest rectangle enclosing the character's shape, described in terms of CHARINFO components, is a rectangle with its upper-left corner at

[x + left-side-bearing, y – ascent]

with a width of

right-side-bearing – left-side-bearing

and a height of

ascent + descent

and the origin for the next character is defined to be

[x + character-width, y]

Note that the baseline is logically viewed as being just below non-descending characters (when descent is zero, only pixels with Y-coordinates less than y are drawn) and that the origin is logically viewed as being coincident with the left edge of a non-kerned character (when left-side-bearing is zero, no pixels with X-coordinate less than x are drawn).

Note that CHARINFO metric values can be negative.

A non-existent character is represented with all CHARINFO components zero.

The interpretation of the per-character attributes field is server-dependent.

QueryTextExtents

font: FONTABLE
string: STRING16

=>

draw-direction: {LeftToRight, RightToLeft}
font-ascent: INT16
font-descent: INT16
overall-ascent: INT16
overall-descent: INT16
overall-width: INT32
overall-left: INT32
overall-right: INT32

Errors: Font

This request returns the logical extents of the specified string of characters in the specified font. If a gcontext is given, the currently contained font is used. The draw-direction, font-ascent, and font-descent are as described in QueryFont. The overall-ascent is the maximum of the ascent metrics of all characters in the string, and overall-descent is the maximum of the descent metrics. The overall-width is the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string, let L be the left-side-bearing metric of the character plus W, and let R be the right-side-bearing metric of the character plus W. The overall-left is the minimum L of all characters in the string, and overall-right is the maximum R.

For fonts defined with linear indexing rather than 2-byte matrix indexing, the server will interpret each CHAR2B as a 16-bit number that has been transmitted most significant byte first (that is, byte1 of the CHAR2B is taken as the most significant byte).

If the font has no defined default-char, then undefined characters in the string are taken to have all zero metrics.

ListFonts

pattern: STRING8
max-names: CARD16

=>

names: LISTofSTRING8

This request returns a list of length at most max-names, of names of available fonts (as controlled by the font search path, see SetFontPath) that match the pattern. The pattern should use the ISO Latin-1 encoding, and upper/lower case does not matter. In the pattern, the "?" character (octal value 77) will match any single character, and the character "*" (octal value 52) will match any number of characters. The returned names are in lowercase.

ListFontsWithInfo

pattern: STRING8
max-names: CARD16

=>+

name: STRING8
info: FONTINFO
replies-hint: CARD32

where

FONTINFO: <same type definition as in QueryFont>

Like ListFonts, but this request also returns information about each font. The information returned for each font is identical to what QueryFont would return, except that the per-character metrics are not returned. Note that this request can generate multiple replies. With each reply, replies-hint may provide an indication of how many more fonts will be returned. This number is a hint only and may be larger or smaller than the number of fonts actually returned. A zero value does not guarantee that no more fonts will be returned. After the font replies, a reply with a zero-length name is sent to indicate the end of the reply sequence.

SetFontPath

path: LISTofSTRING8

Errors: Value

This request defines the search path for font lookup. There is only one search path per server, not one per client. The interpretation of the strings is operating system dependent, but they are intended to specify directories to be searched in the order listed.

Setting the path to the empty list restores the default path defined for the server.

As a side-effect of executing this request, the server is guaranteed to flush all cached information about fonts for which there currently are no explicit resource IDs allocated.

The meaning of an error from this request is system specific.

GetFontPath

=>

path: LISTofSTRING8

This request returns the current search path for fonts.

CreatePixmap

pid: PIXMAP
drawable: DRAWABLE
depth: CARD8
width, height: CARD16

Errors: IDChoice, Drawable, Value, Alloc

This request creates a pixmap and assigns the identifier pid to it. The width and height must be non-zero (else a Value error). The depth must be one of the depths supported by root of the specified drawable (else a Value error). The initial contents of the pixmap are undefined.

It is legal to pass an InputOnly window as a drawable to this request.

FreePixmap

pixmap: PIXMAP

Errors: Pixmap

This request deletes the association between the resource ID and the pixmap. The pixmap storage will be freed when no other resource references it.

CreateGC

cid: GCONTEXT
drawable: DRAWABLE
value-mask: BITMASK
value-list: LISTofVALUE

Errors: IDChoice, Drawable, Pixmap, Font, Match, Value, Alloc

This request creates a graphics context and assigns the identifier cid to it. The gcontext can be used with any destination drawable having the same root and depth as the specified drawable; use with other drawables results in a Match error.

The value-mask and value-list specify which components are to be explicitly initialized. The context components are:

Component.	Type
function	{Clear, And, AndReverse, Copy, AndInverted, Noop, Xor, Or, Nor, Equiv, Invert, OrReverse, CopyInverted, OrInverted, Nand, Set}
plane-mask	CARD32
foreground	CARD32
background	CARD32
line-width	CARD16
line-style	{Solid, OnOffDash, DoubleDash}
cap-style	{NotLast, Butt, Round, Projecting}
join-style	{Miter, Round, Bevel}
fill-style	{Solid, Tiled, OpaqueStippled, Stippled}
fill-rule	{EvenOdd, Winding}
arc-mode	{Chord, PieSlice}
tile	PIXMAP
stipple	PIXMAP
tile-stipple-x-origin	INT16
tile-stipple-y-origin	INT16
font	FONT

Component	Type
subwindow-mode	{ClipByChildren, IncludeInferiors}
graphics-exposures	BOOL
clip-x-origin	INT16
clip-y-origin	INT16
clip-mask	PIXMAP or None
dash-offset	CARD16
dashes	CARD8

In graphics operations, given a source and destination pixel, the result is computed bitwise on corresponding bits of the pixels. That is, a boolean operation is performed in each bit plane. The plane-mask restricts the operation to a subset of planes. That is, the result is:

((src FUNC dst) AND plane-mask) OR (dst AND (NOT plane-mask))

Range checking is not performed on the values for foreground, background, or plane-mask. They are simply truncated to the appropriate number of bits.

The meanings of the functions are:

Function	Operation
Clear	0
And	src AND dst
AndReverse	src AND (NOT dst)
Copy	src
AndInverted	(NOT src) AND dst
NoOp	dst
Xor	src XOR dst
Or	src OR dst
Nor	(NOT src) AND (NOT dst)
Equiv	(NOT src) XOR dst
Invert	NOT dst
OrReverse	src OR (NOT dst)
CopyInverted	NOT src
OrInverted	(NOT src) OR dst
NAnd	(NOT src) OR (NOT dst)
Set	1

The line-width is measured in pixels and can be greater than or equal to one (a "wide" line) or the special value zero (a "thin" line).

Wide lines are drawn centered on the path described by the graphics request. Unless otherwise specified by the join or cap style, the bounding box of a wide line with endpoints [x1, y1], [x2, y2], and width w is a rectangle with vertices at the following real coordinates:

[x1-(w*sn/2), y1+(w*cs/2)], [x1+(w*sn/2), y1-(w*cs/2)],
[x2-(w*sn/2), y2+(w*cs/2)], [x2+(w*sn/2), y2-(w*cs/2)]

The sn is the sine of the angle of the line and cs is the cosine of the angle of the line. A pixel is part of the line (and hence drawn) if the center of the pixel is fully inside the bounding box (which is viewed as having infinitely thin edges). If the center of the pixel is exactly on the bounding box, it is part of the line if and only if the interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are part of the line if and only if the interior is immediately below (y increasing direction). Note that this description is a mathematical model describing the pixels that are drawn for a wide line and does not imply that trigonometry is required to implement such a model. Real or fixed point arithmetic is recommended for computing the corners of the line endpoints for lines greater than one pixel in width.

Thin lines (zero line-width) are "one pixel wide" lines drawn using an unspecified, device dependent algorithm. There are only two constraints on this algorithm. First, if a line is drawn unclipped from [x1,y1] to [x2,y2] and another line is drawn unclipped from [x1+dx,y1+dy] to [x2+dx,y2+dy], then a point [x,y] is touched by drawing the first line if and only if the point [x+dx,y+dy] is touched by drawing the second line. Second, the effective set of points comprising a line cannot be affected by clipping. That is, a point is touched in a clipped line if and only if the point lies inside the clipping region and the point would be touched by the line when drawn unclipped.

Note that a wide line drawn from [x1,y1] to [x2,y2] always draws the same pixels as a wide line drawn from [x2,y2] to [x1,y1], not counting cap and join styles. Implementors are encouraged to make this property true for thin lines, but it is not required. A line-width of zero differs from a line-width of one in which pixels are drawn. In general, drawing a thin line will be faster than drawing a wide line of width one, but thin lines may not mix well aesthetically with wide lines because of the different drawing algorithms. If it is desirable to obtain precise and uniform results across all displays, a client should always use a line-width of one, rather than a line-width of zero.

The line-style defines which sections of a line are drawn:

Solid	The full path of the line is drawn.
DoubleDash	The full path of the line is drawn, but the even dashes are filled differently than the odd dashes (see fill-style), with Butt cap-style used where even and odd dashes meet.
OnOffDash	Only the even dashes are drawn, and cap-style applies to all internal ends of the individual dashes (except NotLast is treated as Butt).

The cap-style defines how the endpoints of a path are drawn:

NotLast	Equivalent to Butt, except that for a line-width of zero or one the final endpoint is not drawn.
Butt	Square at the endpoint (perpendicular to the slope of the line), with no projection beyond.
Round	A circular arc with diameter equal to the line-width, centered on the endpoint; equivalent to Butt for line-width zero or one.
Projecting	Square at the end, but the path continues beyond the endpoint for a distance equal to half the line-width; equivalent to Butt for line-width zero or one.

The join-style defines how corners are drawn for wide lines:

Miter	The outer edges of the two lines extend to meet at an angle.
Round	A circular arc with diameter equal to the line-width, centered on the joinpoint.

Bevel	Butt endpoint styles, and then the triangular "notch" filled.

For a line with coincident endpoints (x1=x2, y1=y2), when the cap-style is applied to both endpoints, the semantics depends on the line-width and the cap-style:

NotLast	thin	Device dependent, but the desired effect is that nothing is drawn.
Butt	thin	Device dependent, but the desired effect is that a single pixel is drawn.
Round	thin	Same as Butt/thin.
Projecting	thin	Same as Butt/thin.
Butt	wide	Nothing is drawn.
Round	wide	The closed path is a circle, centered at the endpoint, with diameter equal to the line-width.
Projecting	wide	The closed path is a square, aligned with the coordinate axes, centered at the endpoint, with sides equal to the line-width.

For a line with coincident endpoints (x1=x2, y1=y2), when the join-style is applied at one or both endpoints, the effect is as if the line was removed from the overall path. However, if the total path consists of (or is reduced to) a single point joined with itself, the effect is the same as when the cap-style is applied at both endpoints.

The tile/stipple and clip origins are interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

The tile pixmap must have the same root and depth as the gcontext (else a Match error). The stipple pixmap must have depth one and must have the same root as the gcontext (else a Match error). For fill-style Stippled (but not fill-style OpaqueStippled), the stipple pattern is tiled in a single plane and acts as an additional clip mask to be ANDed with the clip-mask. Any size pixmap can be used for tiling or stippling, although some sizes may be faster to use than others.

The fill-style defines the contents of the source for line, text, and fill requests. For all text and fill requests (for example, PolyText8, PolyText16, PolyFillRectangle, FillPoly, and PolyFillArc) as well as for line requests (for example, PolyLine, PolySegment, PolyRectangle, PolyArc) with line-style Solid, and for the even dashes for line requests with line-style OnOffDash or DoubleDash:

Solid	Foreground.
Tiled	Tile.
OpaqueStippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one.
Stippled	Foreground masked by stipple.

For the odd dashes for line requests with line-style DoubleDash:

Solid	Background.
Tiled	Same as for even dashes.
OpaqueStippled	Same as for even dashes.

Stippled	Background masked by stipple.

The dashes value allowed here is actually a simplified form of the more general patterns that can be set with SetDashes. Specifying a value of N here is equivalent to specifying the two element list [N, N] in SetDashes. The value must be non-zero (else a Value error). The meaning of dash-offset and dashes are explained in the SetDashes request.

The clip-mask restricts writes to the destination drawable. Only pixels where the clip-mask has a one bit are drawn. Pixels are not drawn outside the area covered by the clip-mask or where the clip-mask has a zero bit. The clip-mask affects all graphics requests, but it does not clip sources. The clip-mask origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. If a pixmap is specified as the clip-mask, it must have depth one and have the same root as the gcontext (else a Match error). If clip-mask is None, then pixels are always drawn (regardless of the clip origin). The clip-mask can also be set with the SetClipRectangles request.

For ClipByChildren, both source and destination windows are additionally clipped by all viewable InputOutput children. For IncludeInferiors, neither source nor destination window is clipped by inferiors. This will result in including subwindow contents in the source and drawing through subwindow boundaries of the destination. The use of IncludeInferiors with a (source or destination) window of one depth with mapped inferiors of differing depth is not illegal, but the semantics is undefined by the core protocol.

The fill-rule defines what pixels are inside (that is, are drawn) for paths given in FillPoly requests. EvenOdd means a point is inside if an infinite ray with the point as origin crosses the path an odd number of times. For Winding, a point is inside if an infinite ray with the point as origin crosses an unequal number of clockwise and counter-clockwise directed path segments. A clockwise directed path segment is one which crosses the ray from left to right as observed from the point. A counter-clockwise segment is one which crosses the ray from right to left as observed from the point. The case where a directed line segment is coincident with the ray is uninteresting because one can simply choose a different ray which is not coincident with a segment.

For both fill rules, a "point" is infinitely small, and the path is an infinitely thin line. A pixel is inside if the center point of the pixel is inside and the center point is not on the boundary. If the center point is on the boundary, the pixel is inside if and only if the polygon interior is immediately to its right (x increasing direction). Pixels with centers along a horizontal edge are a special case and are inside if and only if the polygon interior is immediately below (y increasing direction).

The arc-mode controls filling in the PolyFillArc request.

The graphics-exposures flag controls GraphicsExposure event generation for CopyArea and CopyPlane requests (and any similar requests defined by extensions).

The default component values are:

Component	Default
function	Copy
plane-mask	all ones
foreground	0
background	1
line-width	0
line-style	Solid
cap-style	Butt
join-style	Miter
fill-style	Solid

Component	Default
fill-rule	EvenOdd
arc-mode	PieSlice
tile	Pixmap of unspecified size filled with foreground pixel (that is, client specified pixel if any, else 0) (subsequent changes to foreground do not affect this pixmap)
stipple	Pixmap of unspecified size filled with ones
tile-stipple-x-origin	0
tile-stipple-y-origin	0
font	<server-dependent-font>
subwindow-mode	ClipByChildren
graphics-exposures	True
clip-x-origin	0
clip-y-origin	0
clip-mask	None
dash-offset	0
dashes	4 (that is, the list [4, 4])

Storing a pixmap in a gcontext might or might not result in a copy being made. If the pixmap is later used as the destination for a graphics request, the change might or might not be reflected in the gcontext. If the pixmap is used simultaneously in a graphics request as both a destination and as a tile or stipple, the results are not defined.

It is quite likely that some amount of gcontext information will be cached in display hardware and that such hardware can only cache a small number of gcontexts. Given the number and complexity of components, clients should view switching between gcontexts with nearly identical state as significantly more expensive than making minor changes to a single gcontext.

ChangeGC

gc: GCONTEXT
value-mask: BITMASK
value-list: LISTofVALUE

Errors: GContext, Pixmap, Font, Match, Value, Alloc

This request changes components in gc. The value-mask and value-list specify which components are to be changed. The values and restrictions are the same as for CreateGC.

Changing the clip-mask also overrides any previous SetClipRectangles request on the context. Changing dash-offset or dashes overrides any previous SetDashes request on the context.

The order in which components are verified and altered is server-dependent. If an error is generated, a subset of the components may have been altered.

CopyGC

src-gc, dst-gc: GCONTEXT
value-mask: BITMASK

Errors: GContext, Value, Match, Alloc

This request copies components from src-gc to dst-gc. The value-mask specifies which components to copy, as for CreateGC. The two gcontexts must have the same root and the same depth (else a Match error).

SetDashes

gc: GCONTEXT
dash-offset: CARD16

dashes: LISTofCARD8

Errors: GContext, Value, Alloc

This request sets dash-offset and dashes in gc for dashed line styles. Dashes cannot be empty (else a Value error). Specifying an odd-length list is equivalent to specifying the same list concatenated with itself to produce an even-length list. The initial and alternating elements of dashes are the "even" dashes; the others are the "odd" dashes. All of the elements must be non-zero (else a Value error). The dash-offset defines the phase of the pattern, specifying how many pixels into dashes the pattern should actually begin in any single graphics request. Dashing is continuous through path elements combined with a join-style, but it is reset to the dash-offset each time a cap-style is applied at a line endpoint.

The unit of measure for dashes is the same as in the ordinary coordinate system. Ideally, a dash length is measured along the slope of the line, but implementations are only required to match this ideal for horizontal and vertical lines. Failing the ideal semantics, it is suggested that the length be measured along the major axis of the line. The major axis is defined as the x axis for lines drawn at an angle of between –45 and +45 degrees or between 315 and 225 degrees from the x axis. For all other lines, the major axis is the y axis.

SetClipRectangles

gc: GCONTEXT
clip-x-origin, clip-y-origin: INT16
rectangles: LISTofRECTANGLE
ordering: {UnSorted, YSorted, YXSorted, YXBanded}

Errors: GContext, Value, Alloc, Match

This request changes clip-mask in gc to the specified list of rectangles and sets the clip origin. Output will be clipped to remain contained within the rectangles. The clip origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The rectangle coordinates are interpreted relative to the clip origin. The rectangles should be non-intersecting, or graphics results will be undefined. Note that the list of rectangles can be empty, which effectively disables output. This is the opposite of passing None as the clip-mask in CreateGC and ChangeGC.

If known by the client, ordering relations on the rectangles can be specified with the ordering argument. This may provide faster operation by the server. If an incorrect ordering is specified, the server may generate a Match error, but it is not required to do so. If no error is generated, the graphics results are undefined. UnSorted means the rectangles are in arbitrary order. YSorted means that the rectangles are non-decreasing in their Y origin. YXSorted additionally constrains YSorted order in that all rectangles with an equal Y origin are non-decreasing in their X origin. YXBanded additionally constrains YXSorted by requiring that, for every possible Y scanline, all rectangles that include that scanline have identical Y origins and Y extents.

FreeGC

gc: GCONTEXT

Errors: GContext

This request deletes the association between the resource ID and the gcontext and destroys the gcontext.

ClearArea

window: WINDOW
x, y: INT16
width, height: CARD16
exposures: BOOL

Errors: Window, Value, Match

The x and y coordinates are relative to the window's origin and specify the upper-left corner of the rectangle. If width is zero, it is replaced with the current width of the window minus x. If height is zero, it is replaced with the current height of the window minus y. If the window has a defined background tile, the rectangle is tiled with a plane-mask of all ones and function of Copy. 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.

It is a Match error to use an InputOnly window in this request.

CopyArea

src-drawable, dst-drawable: DRAWABLE
gc: GCONTEXT
src-x, src-y: INT16
width, height: CARD16
dst-x, dst-y: INT16

Errors: Drawable, GContext, Match

This request combines the specified rectangle of src-drawable with the specified rectangle of dst-drawable. The src-x and src-y coordinates are relative to src-drawable's origin. The dst-x and dst-y are relative to dst-drawable's origin, each pair specifying the upper-left corner of the rectangle. The src-drawable must have the same root and the same depth as dst-drawable (else a Match error).

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, but the following occurs on all corresponding destination regions that are either visible or are retained in backing-store. If the dst-drawable is a window with a background other than None, these corresponding destination regions are tiled (with plane-mask of all ones and function Copy) with that background. Regardless of tiling and whether the destination is a window or a pixmap, if graphics-exposures in gc is True, then GraphicsExposure events for all corresponding destination regions are generated.

If graphics-exposures is True but no regions are exposed, then a NoExposure event is generated.

GC components: function, plane-mask, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, clip-mask.

CopyPlane

src-drawable, dst-drawable: DRAWABLE
gc: GCONTEXT
src-x, src-y: INT16
width, height: CARD16
dst-x, dst-y: INT16
bit-plane: CARD32

Errors: Drawable, GContext, Value, Match

The src-drawable must have the same root as dst-drawable (else a Match error), but it need not have the same depth. The bit-plane must have exactly one bit set (else a Value error). Effectively, a pixmap of the same depth as dst-drawable and with size specified by the source region is formed used the foreground/background pixels in gc (foreground everywhere the bit-plane in src-drawable contains a one bit, background everywhere the bit-plane contains a zero bit), and the equivalent of a CopyArea is performed, with all the same exposure semantics. This can also be thought of as using the specified region of the source bit-plane as a stipple with a fill-style of OpaqueStippled for filling a rectangular area of the destination.

GC components: function, plane-mask, foreground, background, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, clip-mask.

PolyPoint

drawable: DRAWABLE
gc: GCONTEXT
coordinate-mode: {Origin, Previous}
points: LISTofPOINT

Errors: Drawable, GContext, Value, Match

This request combines the foreground pixel in gc with the pixel at each point in the drawable. The points are drawn in the order listed.

The first point is always relative to the drawable's origin. The rest are relative either to that origin or the previous point, depending on the coordinate-mode.

GC components: function, plane-mask, foreground, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

PolyLine

drawable: DRAWABLE
gc: GCONTEXT
coordinate-mode: {Origin, Previous}
points: LISTofPOINT

Errors: Drawable, GContext, Value, Match

This request draws lines between each pair of points (point[i], point[i+1]). The lines are drawn in the order listed. The lines join correctly at all intermediate points, and if the first and last points coincide, the first and last lines also join correctly.

For any given line, no pixel is drawn more than once. If thin (zero line-width) lines intersect, the intersecting pixels are drawn multiple times. If wide lines intersect, the intersecting pixels are drawn only once, as though the entire PolyLine were a single filled shape.

The first point is always relative to the drawable's origin. The rest are relative either to that origin or the previous point, depending on the coordinate-mode.

GC components: function, plane-mask, line-width, line-style, cap-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, dashes.

PolySegment

drawable: DRAWABLE
gc: GCONTEXT
segments: LISTofSEGMENT

where

SEGMENT: [x1, y1, x2, y2: INT16]

Errors: Drawable, GContext, Match

For each segment, this request draws a line between [x1, y1] and [x2, y2]. The lines are drawn in the order listed. No joining is performed at coincident end points. For any given line, no pixel is drawn more than once. If lines intersect, the intersecting pixels are drawn multiple times.

GC components: function, plane-mask, line-width, line-style, cap-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, dashes.

PolyRectangle

drawable: DRAWABLE
gc: GCONTEXT
rectangles: LISTofRECTANGLE

Errors: Drawable, GContext, Match

This request draws the outlines of the specified rectangles, as if a five-point PolyLine were specified for each rectangle:

[x,y] [x+width,y] [x+width,y+height] [x,y+height] [x,y]

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 are drawn multiple times.

GC components: function, plane-mask, line-width, line-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, dashes.

PolyArc

drawable: DRAWABLE
gc: GCONTEXT
arcs: LISTofARC

Errors: Drawable, GContext, Match

This request draws circular or elliptical arcs. Each arc is specified by a rectangle and two angles. The angles are signed integers in degrees scaled by 64, with positive indicating counter-clockwise motion and negative indicating clockwise motion. The start of the arc is specified by angle1 relative to the three-oclock position from the center of the rectangle, and the path and extent of the arc is specified by angle2 relative to the start of the arc. If the magnitude of angle2 is greater than 360 degrees, it is truncated to 360 degrees. The x and y coordinates of the rectangle are relative to the origin of the drawable. For an arc specified as [x,y,w,h,a1,a2], the origin of the major and minor axes is at [x+(w/2),y+(h/2)], and the infinitely thin path describing the entire circle/ellipse intersects the horizontal axis at [x,y+(h/2)] and [x+w,y+(h/2)] and intersects the vertical axis at [x+(w/2),y] and [x+(w/2),y+h]. 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 lw, the bounding outlines for filling are given by the infinitely thin paths describing the arcs:

[x+dx/2,y+dy/2,w-dx,h-dy,a1,a2]

and

[x-lw/2,y-lw/2,w+lw,h+lw,a1,a2]

where

dx=min(lw,w)
dy=min(lw,h)

For an arc specified as [x,y,w,h,a1,a2], 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) * w/h) + adjust

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

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

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.

GC components: function, plane-mask, line-width, line-style, cap-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, dashes.

FillPoly

drawable: DRAWABLE
gc: GCONTEXT
shape: {Complex, Nonconvex, Convex}
coordinate-mode: {Origin, Previous}
points: LISTofPOINT

Errors: Drawable, GContext, Match, Value

This request fills the region closed by the specified path. The path is closed automatically if the last point in the list does not coincide with the first point. No pixel of the region is drawn more than once.

The first point is always relative to the drawable's origin. The rest are relative either to that origin or the previous point, depending on the coordinate-mode.

The shape parameter may be used by the server to improve performance. Complex means the path may self-intersect.

Nonconvex means the path does not self-intersect, but the shape is not wholly convex. If known by the client, specifying Nonconvex over Complex may improve performance. If Nonconvex is specified for a self-intersecting path, the graphics results are undefined.

Convex means the path is wholly convex. If known by the client, specifying Convex can improve performance. If Convex is specified for a path that is not convex, the graphics results are undefined.

GC components: function, plane-mask, fill-style, fill-rule, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin.

PolyFillRectangle

drawable: DRAWABLE
gc: GCONTEXT
rectangles: LISTofRECTANGLE

Errors: Drawable, GContext, Match

This request fills the specified rectangles, as if a four-point FillPoly were specified for each rectangle:

[x,y] [x+width,y] [x+width,y+height] [x,y+height]

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 are drawn multiple times.

GC components: function, plane-mask, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin.

PolyFillArc

drawable: DRAWABLE
gc: GCONTEXT
arcs: LISTofARC

Errors: Drawable, GContext, Match

For each arc, this request fills the region closed by the infinitely thin path described by the specified arc and one or two line segments, depending on the arc-mode. For Chord, the single line segment joining the endpoints of the arc is used. For PieSlice, the two line segments joining the endpoints of the arc with the center point are used. The arcs are as specified in the PolyArc request.

The arcs are filled in the order listed. For any given arc, no pixel is drawn more than once. If regions intersect, the intersecting pixels are drawn multiple times.

GC components: function, plane-mask, fill-style, arc-mode, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin.

PutImage

drawable: DRAWABLE
gc: GCONTEXT
depth: CARD8
width, height: CARD16
dst-x, dst-y: INT16
left-pad: CARD8
format: {Bitmap, XYPixmap, ZPixmap}
data: LISTofBYTE

Errors: Drawable, GContext, Match, Value

This request combines an image with a rectangle of the drawable. The dst-x and dst-y coordinates are relative to the drawable's origin.

If Bitmap format is used, then depth must be one (else a Match error), and the image must be in XYFormat. The foreground pixel in gc defines the source for one bits in the image, and the background pixel defines the source for the zero bits.

For XYPixmap and ZPixmap, the depth must match the depth of drawable (else a Match error). For XYPixmap, the image must be sent in XYFormat. For ZPixmap, the image must be sent in the ZFormat defined for the given depth.

The left-pad must be zero for ZPixmap format (else a Match error). For Bitmap and XYPixmap format, left-pad must be less than bitmap-scanline-pad as given in the server connection setup information (else a Match error). The first left-pad bits in every scanline are to be ignored by the server. The actual image begins that many bits into the data. The width argument defines the width of the actual image and does not include left-pad.

GC components: function, plane-mask, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

GC mode-dependent components: foreground, background.

GetImage

drawable: DRAWABLE
x, *y*: INT16
width, *height*: CARD16
plane-mask: CARD32
format: {XYPixmap, ZPixmap}

=>

depth: CARD8
visual: VISUALID or None
data: LISTofBYTE

Errors: Drawable, Value, Match

This request returns the contents of the given rectangle of the drawable in the given format. The x and y coordinates are relative to the drawable's origin and define the upper-left corner of the rectangle. If XYPixmap is specified, only the bit planes specified in plane-mask are transmitted, with the planes appearing from most to least significant in bit order. If ZPixmap is specified, then bits in all planes not specified in plane-mask transmitted as zero. Range checking is not performed on plane-mask; extraneous bits are simply ignored. The returned depth is as specified when the drawable was created and is the same as a depth component in a FORMAT structure (in the connection setup), not a bits-per-pixel component. If the drawable is a window, its visual type is returned. If the drawable is a pixmap, the visual is None.

If the drawable is a pixmap, then the given rectangle must be wholly contained within the pixmap (else a Match error). If the drawable is a window, the window must be mapped, and it must be the case that, if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen and wholly contained within the outside edges of the window (else a Match error). Note that the borders of the window can be included and read with this request. If the window has a backing-store, then the backing-store contents are returned for regions of the window that are obscured by non-inferior windows, but otherwise the returned 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.

This request is not "general purpose" in the same sense as other graphics-related requests. It is intended specifically for rudimentary hardcopy support.

PolyText8

drawable: DRAWABLE
gc: GCONTEXT
x, *y*: INT16
items: LISTofTEXTITEM8

where

TEXTITEM8:	TEXTELT8 or FONT
TEXTELT8:	[delta: INT8

string: STRING8]

Errors: Drawable, GContext, Match, Font

The x and y coordinates are relative to drawable's origin and specify the baseline starting position (the initial character origin). Each text item is processed in turn. A font item causes the font to be stored in gc and to be used for subsequent text. Switching among fonts does not affect the next character origin. A text element delta specifies an additional change in the position along the x axis before the string is drawn; the delta is always added to the character origin. Each character image, as defined by the font in gc, is treated as an additional mask for a fill operation on the drawable.

All contained FONTs are always transmitted most significant byte first.

If a Font error is generated for an item, the previous items may have been drawn.

For fonts defined with two-byte matrix indexing, each STRING8 byte is interpreted as a byte2 value of a CHAR2B with a byte1 value of zero.

GC components: function, plane-mask, fill-style, font, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin.

PolyText16

drawable: DRAWABLE
gc: GCONTEXT
x, *y*: INT16
items: LISTofTEXTITEM16

where

TEXTITEM16:	TEXTELT16 or FONT
TEXTELT16:	[delta: INT8
	string: STRING16]

Errors: Drawable, GContext, Match, Font

Just like PolyText8, except 2-byte (or 16-bit) characters are used. For fonts defined with linear indexing rather than two-byte matrix indexing, the server will interpret each CHAR2B as a 16-bit number that has been transmitted most significant byte first (that is, byte1 of the CHAR2B is taken as the most significant byte).

ImageText8

drawable: DRAWABLE
gc: GCONTEXT
x, *y*: INT16
string: STRING8

Errors: Drawable, GContext, Match

The x and y coordinates are relative to drawable's origin and specify the baseline starting position (the initial character origin). The effect is first to fill a destination rectangle with the background pixel defined in gc and then to paint the text with the foreground pixel. The upper-left corner of the filled rectangle is at

[x, y – font-ascent]

the width is

overall-width

and the height is

font-ascent + font-descent

The overall-width, font-ascent, and font-descent are as they would be returned by a QueryTextExtents call using gc and string.

The function and fill-style defined in gc are ignored for this request. The effective function is Copy and the effective fill-style Solid.

For fonts defined with 2-byte matrix indexing, each STRING8 byte is interpreted as a byte2 value of a CHAR2B with a byte1 value of zero.

GC components: plane-mask, foreground, background, font, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask.

ImageText16

drawable: DRAWABLE
gc: GCONTEXT
x, *y*: INT16
string: STRING16

Errors: Drawable, GContext, Match

Just like ImageText8, except 2-byte (or 16-bit) characters are used. For fonts defined with linear indexing rather than 2-byte matrix indexing, the server will interpret each CHAR2B as a 16-bit number that has been transmitted most significant byte first (that is, byte1 of the CHAR2B is taken as the most significant byte).

CreateColormap

mid: COLORMAP
visual: VISUALID
window: WINDOW
alloc: {None, All}

Errors: IDChoice, Window, Value, Match, Alloc

This request creates a colormap of the specified visual type for the screen on which the window resides and associates the identifier mid with it. The visual type must be one supported by the screen (else a Match error). The initial values of the colormap entries are undefined for classes GrayScale, PseudoColor, and DirectColor. For StaticGray, StaticColor, and TrueColor, the entries will have defined values, but those values are specific to the visual and are not defined by the core protocol. For StaticGray, StaticColor, and TrueColor, alloc must be specified as None (else a Match error). For the other classes, if alloc is None, the colormap initially has no allocated entries, and clients can allocate entries.

If alloc is All, then the entire colormap is "allocated" writable. The initial values of all allocated entries are undefined. For GrayScale and PseudoColor, the effect is as if an AllocColor request returned all pixel values from zero to N-1, where N is the colormap-entries value in the specified visual. For DirectColor, the effect is as if an AllocColorPlanes request returned a pixel value of zero and red-mask, green-mask, and blue-mask values containing the same bits as the corresponding masks in the specified visual. However, in all cases, none of these entries can be freed with FreeColors.

FreeColormap

cmap: COLORMAP

Errors: Colormap

This request deletes the association between the resource ID and the colormap. If the colormap is an installed map for a screen, it is uninstalled (see UninstallColormap). If the colormap is defined as the colormap for a window (by means of CreateWindow or

ChangeWindowAttributes), the colormap for the window is changed to None, and a ColormapNotify event is generated. The colors displayed for a window with a colormap of None are not defined by the protocol.

This request has no effect on a default colormap for a screen.

CopyColormapAndFree

mid, src-cmap: COLORMAP

Errors: IDChoice, Colormap, Alloc

This request creates a colormap of the same visual type and for the same screen as src-cmap, and it associates identifier mid with it. It also moves all of the client's existing allocations from src-cmap to the new colormap with their color values intact and their read-only or writable characteristics intact, and it frees those entries in src-cmap. Color values in other entries in the new colormap are undefined. If src-cmap was created by the client with alloc All (see CreateColormap), then the new colormap is also created with alloc All, all color values for all entries are copied from src-cmap, and then all entries in src-cmap are freed. If src-cmap was not created by the client with alloc All, then the allocations to be moved are all those pixels and planes that have been allocated by the client using either AllocColor, AllocNamedColor, AllocColorCells, or AllocColorPlanes, and which have not been freed since they were allocated.

InstallColormap

cmap: COLORMAP

Errors: Colormap

This request makes this colormap an installed map for its screen. All windows associated with this colormap immediately display with true colors. As a side-effect, additional colormaps might be implicitly installed or uninstalled by the server. Which other colormaps get installed or uninstalled is server-dependent, except that the "required list" must remain installed (see below).

If cmap is not already an installed map, a ColormapNotify event is generated on every window having cmap as an attribute. In addition, for every other colormap that is installed or uninstalled as a result of the request, a ColormapNotify event is generated on every window having that colormap as an attribute.

At any time, there is a subset of the installed maps that are viewed as an ordered list and called the "required list". The length of the required list is at most M, where M is the min-installed-maps specified for the screen in the connection setup. The required list is maintained as follows. When a colormap is an explicit argument to InstallColormap, it is added to the head of the list, and the list is truncated at the tail, if necessary, to keep the length of the list to at most M. When a colormap is an explicit argument to UninstallColormap and it is in the required list, it is removed from the list. A colormap is not added to the required list when it is installed implicitly by the server, and the server cannot implicitly uninstall a colormap that is in the required list.

Initially the default colormap for a screen is installed (but is not in the required list).

UninstallColormap

cmap: COLORMAP

Errors: Colormap

If cmap is on the "required list" for its screen (see InstallColormap), it is removed from the list. As a side-effect, cmap might be uninstalled, and additional colormaps might be implicitly installed or uninstalled. Which colormaps get installed or uninstalled is server-dependent, except that the "required list" must remain installed.

If cmap becomes uninstalled, a ColormapNotify event is generated on every window having cmap as an attribute. In addition, for every other colormap that is installed or uninstalled as a result of the request, a ColormapNotify event is generated on every window having that colormap as an attribute.

ListInstalledColormaps

window: WINDOW

=>

cmaps: LISTofCOLORMAP

Errors: Window

This request returns a list of the currently installed colormaps for the screen of the specified window. The order of colormaps is not significant, and there is no explicit indication of the "required list" (see InstallColormap).

AllocColor

cmap: COLORMAP
red, green, blue: CARD16

=>

pixel: CARD32
red, green, blue: CARD16

Errors: Colormap, Alloc

This request allocates a read-only colormap entry corresponding to the closest RGB values provided by the hardware. It also returns the pixel and the RGB values actually used.

AllocNamedColor

cmap: COLORMAP
name: STRING8

=>

pixel: CARD32
exact-red, exact-green, exact-blue: CARD16
visual-red, visual-green, visual-blue: CARD16

Errors: Colormap, Name, Alloc

This request looks up the named color with respect to the screen associated with the colormap. Then, it does an AllocColor on cmap. The name should use the ISO Latin-1 encoding, and upper/lower case does not matter. The exact RGB values specify the "true" values for the color, and the visual values specify the values actually used in the colormap.

AllocColorCells

cmap: COLORMAP
colors, planes: CARD16
contiguous: BOOL

=>

pixels, masks: LISTofCARD32

Errors: Colormap, Value, Alloc

The number of colors must be positive, and the number of planes must be non-negative (else a Value error). If C colors and P planes are requested, then C pixels and P masks are returned. No mask will have any bits in common with any other mask or with any of the pixels. By ORing together masks and pixels, $C*2^P$ distinct pixels can be produced; all of these are allocated writable by the request. For GrayScale or PseudoColor, each mask will have exactly 1-bit, and for DirectColor, each will have exactly 3-bits. If contiguous is True, then if all 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.

AllocColorPlanes

cmap: COLORMAP
colors, *reds*, *greens*, *blues*: CARD16
contiguous: BOOL

=>

pixels: LISTofCARD32
red-mask, green-mask, blue-mask: CARD32

Errors: Colormap, Value, Alloc

The number of colors must be positive, while the reds, greens, and blues must be non-negative (else a Value error). If C colors, R reds, G greens, and B blues are requested, then C pixels are returned, and the masks have R, G, and B bits set, respectively. If contiguous is True, then each mask will have a contiguous set of bits. No mask will have any bits in common with any other mask or with any of the pixels. For DirectColor, each mask will lie within the corresponding pixel subfield. By ORing together subsets of masks with pixels, $C*2^{R+G+B}$ distinct pixels can be produced; all of these are allocated by the request. The initial RGB values of the allocated entries are undefined. In the colormap, there are only $C*2^R$ independent red entries, $C*2^G$ independent green entries, and $C*2^B$ independent blue entries. This is true even for PseudoColor. When the colormap entry for a pixel value is changed using StoreColors or StoreNamedColor, the pixel is decomposed according to the masks and the corresponding independent entries are updated.

FreeColors

cmap: COLORMAP
pixels: LISTofCARD32
plane-mask: CARD32

Errors: Colormap, Access, Value

The plane-mask should not have any bits in common with any of the pixels. The set of all pixels is produced by ORing together subsets of plane-mask with the pixels. The request frees all of these pixels that were allocated by the client (using AllocColor, AllocNamedColor, AllocColorCells, and AllocColorPlanes). Note that freeing an individual pixel obtained from AllocColorPlanes may not actually allow it to be reused until all of its "related" pixels are also freed.

All specified pixels that are allocated by the client in cmap are freed, even if one or more pixels produce an error. A Value error is generated if a specified pixel is not a valid index into cmap, and an Access error is generated if a specified pixel is not allocated by the client (that is, is unallocated or is only allocated by another client). If more than one pixel is in error, which one is reported is arbitrary.

StoreColors

cmap: COLORMAP
items: LISTofCOLORITEM

where

```
COLORITEM:    [pixel: CARD32
               do-red, do-green, do-blue: BOOL
               red, green, blue: CARD16]
```

Errors: Colormap, Access, Value

This request changes the colormap entries of the specified pixels. The do-red, do-green, and do-blue fields indicate which components should actually be changed. If the colormap is an installed map for its screen, the changes are visible immediately.

All specified pixels that are allocated writable in cmap (by any client) are changed, even if one or more pixels produce an error. A Value error is generated if a specified pixel is not a valid index into cmap, and an Access error is generated if a specified pixel is unallocated or is allocated read-only. If more than one pixel is in error, which one is reported is arbitrary.

StoreNamedColor

cmap: COLORMAP
pixel: CARD32
name: STRING8
do-red, do-green, do-blue: BOOL

Errors: Colormap, Name, Access, Value

This request looks up the named color with respect to the screen associated with cmap and then does a StoreColors in cmap. The name should use the ISO Latin-1 encoding, and upper/lower case does not matter. The Access and Value errors are the same as in StoreColors.

QueryColors

cmap: COLORMAP
pixels: LISTofCARD32

=>

colors: LISTofRGB

where

RGB: [red, green, blue: CARD16]

Errors: Colormap, Value

This request returns the color values stored in cmap for the specified pixels. The values returned for an unallocated entry are undefined. A Value error is generated if a pixel is not a valid index into cmap. If more than one pixel is in error, which one is reported is arbitrary.

LookupColor

cmap: COLORMAP
name: STRING8

=>

exact-red, exact-green, exact-blue: CARD16
visual-red, visual-green, visual-blue: CARD16

Errors: Colormap, Name

This request looks up the string name of a color with respect to the screen associated with cmap and returns both the exact the color values and the closest values provided by the hardware with respect to the visual type of cmap. The name should use the ISO Latin-1 encoding, and upper/lower case does not matter.

CreateCursor

cid: CURSOR
source: PIXMAP
mask: PIXMAP or None
fore-red, fore-green, fore-blue: CARD16
back-red, back-green, back-blue: CARD16
x, y: CARD16

Errors: IDChoice, Pixmap, Match, Alloc

This request creates a cursor and associates identifier cid with it. The foreground and background RGB values must be specified, even if the server only has a StaticGray or GrayScale screen. The foreground is used for the one bits in the source, and the background is used for the zero bits. Both source and mask (if specified) must have depth one (else a Match error), but they can have any root. The mask pixmap defines the shape of the cursor. That is, the one bits in the mask define which source pixels will be displayed. Where the mask has zero bits, the corresponding bits of the source pixmap are ignored. If no mask is given, all pixels of the source are displayed. The mask, if present, must be the same size as source (else a Match error). The x and y coordinates define the hotspot, relative to the source's origin, and must be a point within the source (else a Match error).

The components of the cursor may be transformed arbitrarily to meet display limitations.

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

Subsequent drawing in the source or mask pixmap has an undefined effect on the cursor. The server might or might not make a copy of the pixmap.

CreateGlyphCursor

cid: CURSOR
source-font: FONT
mask-font: FONT or None
source-char, mask-char: CARD16
fore-red, fore-green, fore-blue: CARD16
back-red, back-green, back-blue: CARD16

Errors: IDChoice, Font, Value, Alloc

This request is similar to CreateCursor, except the source and mask bitmaps are obtained from the specified font glyphs. The source-char must be a defined glyph in source-font and, if mask-font is given, mask-char must be a defined glyph in mask-font (else a Value error). The mask font and character are optional. The origins of the source and mask (if it is defined) glyphs are positioned coincidently and define the hotspot. The source and mask need not have the same bounding box metrics, and there is no restriction on the placement of the hotspot relative to the bounding boxes. If no mask is given, all pixels of the source are displayed. Note that source-char and mask-char are CARD16 (not CHAR2B). For two-byte matrix fonts, the 16-bit value should be formed with byte1 in the most significant byte and byte2 in the least significant byte.

The components of the cursor may be transformed arbitrarily to meet display limitations.

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

FreeCursor

cursor: CURSOR

Errors: Cursor

This request deletes the association between the resource ID and the cursor. The cursor storage will be freed when no other resource references it.

RecolorCursor

cursor: CURSOR
fore-red, fore-green, fore-blue: CARD16
back-red, back-green, back-blue: CARD16

Errors: Cursor

This request changes the color of a cursor. If the cursor is being displayed on a screen, the change is visible immediately.

QueryBestSize

class: {Cursor, Tile, Stipple}
drawable: DRAWABLE
width, height: CARD16

=>

width, height: CARD16

Errors: Drawable, Value, Match

This request returns the "best" size that is "closest" to the argument size. For Cursor, this is the largest size that can be fully displayed. For Tile, this is the size that can be tiled "fastest". For Stipple, this is the size that can be stippled "fastest".

For Cursor, the drawable indicates the desired screen. For Tile and Stipple, the drawable indicates screen and also possibly window class and depth. An InputOnly window cannot be used as the drawable for Tile or Stipple (else a Match error).

QueryExtension

name: STRING8

=>

present: BOOL
major-opcode: CARD8
first-event: CARD8
first-error: CARD8

This request determines if the named extension is present. If so, the major opcode for the extension is returned, if it has one. Otherwise, zero is returned. Any minor opcode and the request formats are specific to the extension. If the extension involves additional event types, the base event type code is returned. Otherwise, zero is returned. The format of the events is specific to the extension. If the extension involves additional error codes, the base error code is returned. Otherwise, zero is returned. The format of additional data in the errors is specific to the extension.

The extension name should use the ISO Latin-1 encoding, and upper/lower case matters.

ListExtensions

=>

names: LISTofSTRING8

This request returns a list of all extensions supported by the server.

SetModifierMapping

keycodes-per-modifier: CARD8
keycodes: LISTofKEYCODE

=>

status: {Success, Busy, Failed}

Errors: Value, Alloc

This request specifies the keycodes (if any) of the keys to be used as modifiers. The number of keycodes in the list must be 8*keycodes-per-modifier (else a Length error). The keycodes are divided into eight sets, with the sets, with each set containing keycodes-per-modifier elements. The sets are assigned in order to the modifiers Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, and Mod5. Only non-zero keycode values are used within each set; zero values are ignored. All of the non-zero keycodes must be in the range specified by min-keycode and max-keycode in the connection setup (else a Value error). The order of keycodes within a set does not matter. If no non-zero values are specified in a set, the use of the corresponding modifier is disabled, and the modifier bit will always be zero. Otherwise, the modifier bit will be one whenever at least one of the keys in the corresponding set is in the down position.

A server can impose restrictions on how modifiers can be changed (for example, if certain keys do not generate up transitions in hardware or if multiple keys per modifier are not supported). The status reply is Failed if some such restriction is violated, and none of the modifiers are changed.

If the new non-zero keycodes specified for a modifier differ from those currently defined, and any (current or new) keys for that modifier are logically in the down state, then the status reply is Busy, and none of the modifiers are changed.

This request generates a MappingNotify event on a Success status.

GetModifierMapping

=>

keycodes-per-modifier: CARD8
keycodes: LISTofKEYCODE

This request returns the keycodes of the keys being used as modifiers. The number of keycodes in the list is 8*keycodes-per-modifier. The keycodes are divided into eight sets, with each set containing keycodes-per-modifier elements. The sets are assigned in order to the modifiers Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, and Mod5. The keycodes-per-modifier value is chosen arbitrarily by the server; zeroes are used to fill in unused elements within each set. If only zero values are given in a set, the use of the corresponding modifier has been disabled. The order of keycodes within each set is chosen arbitrarily by the server.

ChangeKeyboardMapping

first-keycode: KEYCODE
keysyms-per-keycode: CARD8
keysyms: LISTofKEYSYM

Errors: Value, Alloc

This request defines the symbols for the specified number of keycodes, starting with the specified keycode. The symbols for keycodes outside this range remained unchanged. The number of elements in the keysyms list must be a multiple of keysyms-per-keycode (else a Length error). The first-keycode must be greater than or equal to min-keycode as returned in the connection setup (else a Value error); and

first-keycode + (keysyms-length / keysyms-per-keycode) – 1

must be less than or equal to max-keycode as returned in the connection setup (else a Value error). KEYSYM number N (counting from zero) for keycode K has an index (counting from zero) of

(K – first-keycode) * keysyms-per-keycode + N

in keysyms. The 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 non-trailing positions of the effective list for a keycode.

This request generates a MappingNotify event.

There is no requirement that the server interpret this mapping; it is merely stored for reading and writing by clients (see Section 6).

GetKeyboardMapping

first-keycode: KEYCODE
count: CARD8

=>

keysyms-per-keycode: CARD8
keysyms: LISTofKEYSYM

Errors: Value

This request returns the symbols for the specified number of keycodes, starting with the specified keycode. The first-keycode must be greater than or equal to min-keycode as returned in the connection setup (else a Value error), and

first-keycode + count − 1

must be less than or equal to max-keycode as returned in the connection setup (else a Value error). The number of elements in the keysyms list is

count * keysyms-per-keycode

and KEYSYM number N (counting from zero) for keycode K has an index (counting from zero) of

(K − first-keycode) * keysyms-per-keycode + N

in keysyms. 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.

ChangeKeyboardControl

value-mask: BITMASK
value-list: LISTofVALUE

Errors: Match, Value

This request controls various aspects of the keyboard. The value-mask and value-list specify which controls are to be changed. The possible values are:

Control	Type
key-click-percent	INT8
bell-percent	INT8
bell-pitch	INT16
bell-duration	INT16
led	CARD8
led-mode	{On, Off}
key	KEYCODE
auto-repeat-mode	{On, Off, Default}

The key-click-percent sets the volume for key clicks between 0 (off) and 100 (loud) inclusive, if possible. Setting to −1 restores the default. Other negative values generate a Value error.

The bell-percent sets the base volume for the bell between 0 (off) and 100 (loud) inclusive, if possible. Setting to −1 restores the default. Other negative values generate a Value error.

The bell-pitch sets the pitch (specified in Hz) of the bell, if possible. Setting to −1 restores the default. Other negative values generate a Value error.

The bell-duration sets the duration (specified in milliseconds) of the bell, if possible. Setting to −1 restores the default. Other negative values generate a Value error.

If both led-mode and led are specified, then the state of that LED is changed, if possible. If only led-mode is specified, then the state of all LEDs are changed, if possible. At most 32 LEDs, numbered from one, are supported. No standard interpretation of LEDs is defined. It is a Match error if an led is specified without an led-mode.

If both auto-repeat-mode and key are specified, then the auto-repeat mode of that key is changed, if possible. If only auto-repeat-mode is specified, then the global auto-repeat mode for the entire keyboard is changed, if possible, without affecting the per-key settings. It is a Match error if a key is specified without an auto-repeat-mode.

A bell generator connected with the console but not directly on the keyboard is treated as if it were part of the keyboard.

The order in which controls are verified and altered is server-dependent. If an error is generated, a subset of the controls may have been altered.

GetKeyboardControl

=>

key-click-percent: CARD8
bell-percent: CARD8
bell-pitch: CARD16
bell-duration: CARD16
led-mask: CARD32
global-auto-repeat: { On, Off }
auto-repeats: LISTofCARD8

This request returns the current control values for the keyboard. For the LEDs, the least significant bit of led-mask corresponds to LED one, and each one bit in led-mask indicates an LED that is lit. The auto-repeats is a bit vector; each one bit 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.

Bell

percent: INT8

Errors: Value

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

base – [(base * percent) / 100] + percent

and when percent is negative:

base + [(base * percent) / 100]

SetPointerMapping

map: LISTofCARD8

=>

status: { Success, Busy }

Errors: Value

This request sets the mapping of the pointer. Elements of the list are indexed starting from one. The length of the list must be the same as GetPointerMapping would return (else a Value error). The index is a "core" button number, and the element of the list defines the "effective" number.

A zero element disables a button, and elements are not restricted in value by the number of physical buttons, but no two elements can have the same non-zero value (else a Value error).

If any of the buttons to be altered are logically in the down state, the status reply is Busy, and the mapping is not changed.

This request generates a MappingNotify event on a Success status.

GetPointerMapping

=>

map: LISTofCARD8

This request returns the current mapping of the pointer. Elements of the list are indexed starting from one. The length of the list indicates the number of physical buttons.

The nominal mapping for a pointer is the identity mapping; map[i]=i.

ChangePointerControl

do-acceleration, *do-threshold*: BOOL
acceleration-numerator, *acceleration-denominator*: INT16
threshold: INT16

Errors: Value

This request defines how the pointer moves. The acceleration is a multiplier for movement, expressed as a fraction. For example, specifying 3/1 means the pointer moves three times as fast as normal. The fraction can be rounded arbitrarily by the server. Acceleration only takes effect if the pointer moves more than threshold pixels at once and only applies to the amount beyond the threshold. Setting a value to −1 restores the default. Other negative values generate a Value error, as does a zero value for acceleration-denominator.

GetPointerControl

=>

acceleration-numerator, acceleration-denominator: CARD16
threshold: CARD16

This request returns the current acceleration and threshold for the pointer.

SetScreenSaver

timeout, *interval*: INT16
prefer-blanking: {Yes, No, Default}
allow-exposures: {Yes, No, Default}

Errors: Value

The timeout and interval are specified in seconds; setting a value to -1 restores the default. Other negative values generate a Value error. If the timeout value is zero, screen-saver is disabled. If the timeout value is non-zero, screen-saver is enabled. Once screen-saver is enabled, if no input from the keyboard or pointer is generated for timeout seconds, screen-saver is activated. For each screen, if blanking is preferred and the hardware supports video blanking, the screen will simply go blank. Otherwise, if either exposures are allowed or the screen can be regenerated without sending exposure events to clients, the screen is changed in a server-dependent fashion to avoid phosphor burn. Otherwise, the state of the screens do not change, and screen-saver is not activated. Screen-saver is deactivated, and all screen states are restored, at the next keyboard or pointer input or at the next ForceScreenSaver with mode Reset.

If the server-dependent screen-saver method is amenable to periodic change, interval serves as a hint about how long the change period should be, with zero hinting that no periodic change should be make. Examples of ways to change the screen include scrambling the color map periodically, moving an icon image about the screen periodically, or tiling the screen with the root window background tile, randomly re-origined periodically.

GetScreenSaver

=>

timeout, interval: CARD16
prefer-blanking: {Yes, No}
allow-exposures: {Yes, No}

This request returns the current screen-saver control values.

ForceScreenSaver

mode: { Activate, Reset }

Errors: Value

If the mode is Activate and screen-saver is currently deactivated, then screen-saver is activated (even if screen-saver has been disabled with a timeout value of zero). If the mode is Reset and screen-saver is currently enabled, then screen-saver is deactivated (if it was activated), and then the activation timer is reset to its initial state, as if device input had just been received.

ChangeHosts

mode: { Insert, Delete }
host: HOST

Errors: Access, Value

This request adds or removes the specified host from the access control list. When the access control mechanism is enabled and a host attempts to establish a connection to the server, the host must be in this list, or the server will refuse the connection.

The client must reside on the same host as the server and/or have been granted permission by a server-dependent method to execute this request (else an Access error).

An initial access control list can usually be specified, typically by naming a file that the server reads at startup and reset.

The following address families are defined. A server is not required to support these families and may support families not listed here. Use of an unsupported family, an improper address format, or an improper address length within a supported family results in a Value error.

For the Internet family, the address must be 4 bytes long. The address bytes are in standard IP order; the server performs no automatic swapping on the address bytes. For a Class A address, the network number is the first byte in the address, and the host number is the remaining 3 bytes, most significant byte first. For a Class B address, the network number is the first 2 bytes and the host number is the last 2 bytes, each most significant byte first. For a Class C address, the network number is the first 3 bytes, most significant byte first, and the last byte is the host number.

For the DECnet family, the server performs no automatic swapping on the address bytes. A Phase IV address is 2 bytes long: the first byte contains the least significant 8-bits of the node number, and the second byte contains the most significant 2-bits of the node number in the least significant 2-bits of the byte and the area in the most significant 6-bits of the byte.

For the Chaos family, the address must be 2 bytes long. The host number is always the first byte in the address, and the subnet number is always the second byte. The server performs no automatic swapping on the address bytes.

ListHosts

=>

mode: { Enabled, Disabled }
hosts: LISTofHOST

This request returns the hosts on the access control list and whether use of the list at connection setup is currently enabled or disabled.

Each HOST is padded to a multiple of four bytes.

SetAccessControl

mode: { Enable, Disable }

Errors: Value, Access

This request enables or disables the use of the access control list at connection setups.

The client must reside on the same host as the server and/or have been granted permission by a server-dependent method to execute this request (else an Access error).

SetCloseDownMode

mode: {Destroy, RetainPermanent, RetainTemporary}

Errors: Value

This request defines what will happen to the client's resources at connection close. A connection starts in Destroy mode. The meaning of the close-down mode is described in Section 11.

KillClient

resource: CARD32 or AllTemporary

Errors: Value

If a valid resource is specified, KillClient forces a close-down of the client that created the resource. If the client has already terminated in either RetainPermanent or RetainTemporary mode, all of the client's resources are destroyed (see Section 11). If AllTemporary is specified, then the resources of all clients that have terminated in RetainTemporary are destroyed.

NoOperation

This request has no arguments and no results, but the request length field can be non-zero, which allows the request to be any multiple of 4-bytes in length. The bytes contained in the request are uninterpreted by the server.

This request can be used in its minimum 4 byte form as ''padding'' where necessary by client libraries that find it convenient to force requests to begin on 64-bit boundaries.

11. Connection Close

What happens at connection close?

All event selections made by the client are discarded. If the client has the pointer actively grabbed, an UngrabPointer is performed. If the client has the keyboard actively grabbed, an UngrabKeyboard is performed. All passive grabs by the client are released. If the client has the server grabbed, an UngrabServer is performed. All selections (see SetSelectionOwner) owned by the client are disowned. If close-down mode (see SetCloseDownMode) is RetainPermanent or RetainTemporary, then all resources (including colormap entries) allocated by the client are marked as ''permanent'' or ''temporary'', respectively (but this does not prevent other clients from explicitly destroying them). If the mode is Destroy, then all of the client's resources are destroyed as described below.

What happens when a client's resources are destroyed?

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

What happens when the last connection to a server closes?

A server goes through a cycle of having no connections and having some connections. At every transition to the state of having no connections as a result of a connection closing with a Destroy close-down mode, the server ''resets'' its state, as if it had just been started.

This starts by destroying all lingering resources from clients that have terminated in RetainPermanent or RetainTemporary mode. It additionally includes deleting all but the predefined atom identifiers, deleting all properties on all root windows, resetting all device maps and attributes (key click, bell volume, acceleration), resetting the access control list, restoring the standard root tiles and cursors, restoring the default font path, and restoring the input focus to state PointerRoot.

Note that closing a connection with a close-down mode of RetainPermanent or RetainTemporary will not cause the server to reset.

12. Events

When a button press is processed, with the pointer in some window W, and no active pointer grab is in progress, then the ancestors of W are searched from the root down, looking for a passive grab to activate. If no matching passive grab on the button exists, then an active grab is started automatically for the client receiving the event, and the last-pointer-grab time is set to the current server time. The effect is essentially equivalent to a GrabButton with arguments:

Argument	Value
event-window	event window
event-mask	client's selected pointer events on the event window
pointer-mode and keyboard-mode	Asynchronous
owner-events	True if the client has OwnerGrabButton selected on the event window, else False
confine-to	None
cursor	None

The grab is terminated automatically when the logical state of the pointer has all buttons released. UngrabPointer and ChangeActiveGrab can both be used to modify the active grab.

KeyPress
KeyRelease
ButtonPress
ButtonRelease
MotionNotify

root, event: WINDOW
child: WINDOW or None
same-screen: BOOL
root-x, root-y, event-x, event-y: INT16
detail: <see below>
state: SETofKEYBUTMASK
time: TIMESTAMP

These events are generated either when a key or button logically changes state or when the pointer logically moves. The generation of these logical changes may lag the physical changes, if device event processing is frozen. Note that KeyPress and KeyRelease are generated for all keys, even those mapped to modifier bits. The "source" of the event is the window the pointer is in. The window with respect to which the event is normally reported is found by looking up the hierarchy (starting with the source window) for the first window on which any client has selected interest in the event, provided no intervening window prohibits event generation by including the event type in its do-not-propagate-mask. The actual window used for reporting can be modified by active grabs and, in the case of keyboard events, can be modified by the focus window. The window the event is reported with respect to is called the "event" window.

The root is the root window of the "source" window, and root-x and root-y are the pointer coordinates relative to root's origin at the time of the event. Event is the "event" window. If the event window is on the same screen as root, then event-x and event-y are the pointer coordinates relative to the event window's origin. Otherwise, event-x and event-y are zero. If the source window is an inferior of the event window, then child is set to the child of the event window that is an ancestor of (or is) the source window. Otherwise, it is set to None. The state component gives the logical state of the buttons and modifier keys just before the event. The detail component type varies with the event type:

Event	Component
KeyPress, KeyRelease	KEYCODE
ButtonPress, ButtonRelease	BUTTON
MotionNotify	{Normal, Hint}

MotionNotify events are only generated when the motion begins and ends in the window. The granularity of motion events is not guaranteed, but a client selecting for motion events is guaranteed to get at least one event when the pointer moves and comes to rest. Selecting PointerMotion receives events independent of the state of the pointer buttons. By selecting some subset of Button[1-5]Motion instead, MotionNotify events will only be received when one or more of the specified buttons are pressed. By selecting ButtonMotion, MotionNotify events will be received only when at least one button is pressed. The events are always of type MotionNotify, independent of the selection. If PointerMotionHint is selected, the server is free to send only one MotionNotify event (with detail Hint) to the client for the event window, until either the key or button state changes, or the pointer leaves the event window, or the client issues a QueryPointer or GetMotionEvents request.

EnterNotify
LeaveNotify

root, event: WINDOW
child: WINDOW or None
same-screen: BOOL
root-x, root-y, event-x, event-y: INT16
mode: {Normal, Grab, Ungrab}
detail: {Ancestor, Virtual, Inferior, Nonlinear, NonlinearVirtual}
focus: BOOL
state: SETofKEYBUTMASK
time: TIMESTAMP

If pointer motion or window hierarchy change causes the pointer to be in a different window than before, EnterNotify and LeaveNotify events are generated instead of a MotionNotify event. Only clients selecting EnterWindow on a window receive EnterNotify events, and only clients selection LeaveNotify receive LeaveNotify events. The pointer position reported in the event is always the "final" position, not the "initial" position of the pointer. The root is the root window for this position, and root-x and root-y are the pointer coordinates relative to root's origin at the time of the event. Event is the event window. If the event window is on the same screen as root, then event-x and event-y are the pointer coordinates relative to the event window's origin. Otherwise, event-x and event-y are zero. In a LeaveNotify event, if a child of the event window contains the "initial" position of the pointer, then the child component is set to that child. Otherwise, it is None. For an EnterNotify event, if a child of the event window contains the "final" pointer position, then the child component is set to that child. Otherwise, it is None. If the event window is the focus window or an inferior of the focus window, then focus is True.

Otherwise, focus is False.

Normal pointer motion events have mode Normal. Pseudo-motion events when a grab actives have mode Grab, and pseudo-motion events when a grab deactivates have mode Ungrab.

All EnterNotify and LeaveNotify events caused by a hierarchy change are generated after any hierarchy event (that is, UnmapNotify, MapNotify, ConfigureNotify, GravityNotify, CirculateNotify) caused by that change, but the ordering of EnterNotify and LeaveNotify events with respect to FocusOut, VisibilityNotify, and Expose events is not constrained.

Normal events are generated as follows:

When the pointer moves from window A to window B, and A is an inferior of B:

- LeaveNotify with detail Ancestor is generated on A.
- LeaveNotify with detail Virtual is generated on each window between A and B exclusive (in that order).
- EnterNotify with detail Inferior is generated on B.

When the pointer moves from window A to window B, and B is an inferior of A:

- LeaveNotify with detail Inferior is generated on A.
- EnterNotify with detail Virtual is generated on each window between A and B exclusive (in that order).
- EnterNotify with detail Ancestor is generated on B.

When the pointer moves from window A to window B, with window C being their least common ancestor:

- LeaveNotify with detail Nonlinear is generated on A.
- LeaveNotify with detail NonlinearVirtual is generated on each window between A and C exclusive (in that order).
- EnterNotify with detail NonlinearVirtual is generated on each window between C and B exclusive (in that order).
- EnterNotify with detail Nonlinear is generated on B.

When the pointer moves from window A to window B, on different screens:

- LeaveNotify with detail Nonlinear is generated on A.
- If A is not a root window, LeaveNotify with detail NonlinearVirtual is generated on each window above A up to and including its root (in order).
- If B is not a root window, EnterNotify with detail NonlinearVirtual is generated on each window from B's root down to but not including B (in order).
- EnterNotify with detail Nonlinear is generated on B.

When a pointer grab activates (but after any initial warp into a confine-to window, and before generating any actual ButtonPress event that activates the grab), with G the grab-window for the grab and P the window the pointer is in:

- EnterNotify and LeaveNotify events with mode Grab are generated (as for Normal above) as if the pointer were to suddenly warp from its current position in P to some position in G. However, the pointer does not warp, and the pointer position is used as both the "initial" and "final" positions for the events.

When a pointer grab deactivates (but after generating any actual ButtonRelease event that deactivates the grab), with G the grab-window for the grab and P the window the pointer is in:

- EnterNotify and LeaveNotify events with mode Ungrab are generated (as for Normal above) as if the pointer were to suddenly warp from from some position in G to its current position in P. However, the pointer does not warp, and the current pointer position is used as both the "initial" and "final" positions for the events.

FocusIn
FocusOut

event: WINDOW
mode: {Normal, WhileGrabbed, Grab, Ungrab}
detail: {Ancestor, Virtual, Inferior, Nonlinear, NonlinearVirtual, Pointer, PointerRoot, None}

These events are generated when the input focus changes and are reported to clients selecting FocusChange on the window. Events generated by SetInputFocus when the keyboard is not grabbed have mode Normal. Events generated by SetInputFocus when the keyboard is grabbed have mode WhileGrabbed. Events generated when a keyboard grab actives have mode Grab, and events generated when a keyboard grab deactivates have mode Ungrab.

All FocusOut events caused by a window unmap are generated after any UnmapNotify event, but the ordering of FocusOut with respect to generated EnterNotify, LeaveNotify, VisibilityNotify, and Expose events is not constrained.

Normal and WhileGrabbed events are generated as follows:

When the focus moves from window A to window B, and A is an inferior of B, with the pointer in window P:

- FocusOut with detail Ancestor is generated on A.
- FocusOut with detail Virtual is generated on each window between A and B exclusive (in that order).
- FocusIn with detail Inferior is generated on B.
- If P is an inferior of B, but P is not A or an inferior of A or an ancestor of A, FocusIn with detail Pointer is generated on each window below B down to and including P (in order).

When the focus moves from window A to window B, and B is an inferior of A, with the pointer in window P:

- If P is an inferior of A, but P is not A or an inferior of B or an ancestor of B, FocusOut with detail Pointer is generated on each window from P up to but not including A (in order).
- FocusOut with detail Inferior is generated on A.
- FocusIn with detail Virtual is generated on each window between A and B exclusive (in that order).
- FocusIn with detail Ancestor is generated on B.

When the focus moves from window A to window B, with window C being their least common ancestor, and with the pointer in window P:

- If P is an inferior of A, FocusOut with detail Pointer is generated on each window from P up to but not including A (in order).
- FocusOut with detail Nonlinear is generated on A.
- FocusOut with detail NonlinearVirtual is generated on each window between A and C exclusive (in that order).
- FocusIn with detail NonlinearVirtual is generated on each window between C and B exclusive (in that order).
- FocusIn with detail Nonlinear is generated on B.
- If P is an inferior of B, FocusIn with detail Pointer is generated on each window below B down to and including P (in order).

When the focus moves from window A to window B, on different screens, with the pointer in window P:

- If P is an inferior of A, FocusOut with detail Pointer is generated on each window from P up to but not including A (in order).
- FocusOut with detail Nonlinear is generated on A.
- If A is not a root window, FocusOut with detail NonlinearVirtual is generated on each window above A up to and including its root (in order).
- If B is not a root window, FocusIn with detail NonlinearVirtual is generated on each window from B's root down to but not including B (in order).
- FocusIn with detail Nonlinear is generated on B.
- If P is an inferior of B, FocusIn with detail Pointer is generated on each window below B down to and including P (in order).

When the focus moves from window A to PointerRoot (or None), with the pointer in window P:

- If P is an inferior of A, FocusOut with detail Pointer is generated on each window from P up to but not including A (in order).
- FocusOut with detail Nonlinear is generated on A.
- If A is not a root window, FocusOut with detail NonlinearVirtual is generated on each window above A up to and including its root (in order).
- FocusIn with detail PointerRoot (or None) is generated on all root windows.
- If the new focus is PointerRoot, FocusIn with detail Pointer is generated on each window from P's root down to and including P (in order).

When the focus moves from PointerRoot (or None) to window A, with the pointer in window P:

- If the old focus is PointerRoot, FocusOut with detail Pointer is generated on each window from P up to and including P's root (in order).
- FocusOut with detail PointerRoot (or None) is generated on all root windows.
- If A is not a root window, FocusIn with detail NonlinearVirtual is generated on each window from A's root down to but not including A (in order).
- FocusIn with detail Nonlinear is generated on A.
- If P is an inferior of A, FocusIn with detail Pointer is generated on each window below A down to and including P (in order).

When the focus moves from PointerRoot to None (or vice versa), with the pointer in window P:

- If the old focus is PointerRoot, FocusOut with detail Pointer is generated on each window from P up to and including P's root (in order).
- FocusOut with detail PointerRoot (or None) is generated on all root windows.
- FocusIn with detail None (or PointerRoot) is generated on all root windows.
- If the new focus is PointerRoot, FocusIn with detail Pointer is generated on each window from P's root down to and including P (in order).

When a keyboard grab activates (but before generating any actual KeyPress event that activates the grab), with G the grab-window for the grab and F the current focus:

- FocusIn and FocusOut events with mode Grab are generated (as for Normal above) as if the focus were to change from F to G.

When a keyboard grab deactivates (but after generating any actual KeyRelease event that deactivates the grab), with G the grab-window for the grab and F the current focus:

- FocusIn and FocusOut events with mode Ungrab are generated (as for Normal above) as if the focus were to change from G to F.

KeymapNotify

keys: LISTofCARD8

The value is a bit vector as described in QueryKeymap. This event is reported to clients selecting KeymapState on a window and is generated immediately after every EnterNotify and FocusIn.

Expose

window: WINDOW
x, y, width, height: CARD16
count: CARD16

This event is reported to clients selecting Exposure on the window. It is generated when no valid contents are available for regions of a window, and either a) the regions are visible, b) the regions are viewable and the server is (perhaps newly) maintaining backing store on the window, or c) the window is not viewable but the server is (perhaps newly) honoring window's backing-store attribute of Always or WhenMapped. The regions are decomposed into an (arbitrary) set of rectangles, and an Expose event is generated for each rectangle.

For a given ''action'' causing exposure events, the set of events for a given window are guaranteed to be reported contiguously. If count is zero, then no more Expose events for this window follow. If count is non-zero, then at least that many more Expose events for this window follow (and possibly more).

The x and y coordinates are relative to drawable's origin and specify the upper-left corner of a rectangle. The width and height specify the extent of the rectangle.

Expose events are never generated on InputOnly windows.

All Expose events caused by a hierarchy change are generated after any hierarchy event (for example, UnmapNotify, MapNotify, ConfigureNotify, GravityNotify, CirculateNotify) caused by that change. All Expose events on a given window are generated after any VisibilityNotify event on that window, but it is not required that all Expose events on all windows be generated after all Visibilitity events on all windows. The ordering of Expose events with respect to FocusOut, EnterNotify, and LeaveNotify events is not constrained.

GraphicsExposure

drawable: DRAWABLE
x, y, width, height: CARD16
count: CARD16
major-opcode: CARD8
minor-opcode: CARD16

This event is reported to clients selecting graphics-exposures in a graphics context and is generated when a destination region could not be computed due to an obscured or out-of-bounds source region. All of the regions exposed by a given graphics request are guaranteed to be reported contiguously. If count is zero then no more GraphicsExposure events for this window follow. If count is non-zero, then at least that many more GraphicsExposure events for this window follow (and possibly more).

The x and y coordinates are relative to drawable's origin and specify the upper-left corner of a rectangle. The width and height specify the extent of the rectangle.

The major and minor opcodes identify the graphics request used. For the core protocol, major-opcode is always CopyArea or CopyPlane, and minor-opcode is always zero.

NoExposure

drawable: DRAWABLE
major-opcode: CARD8
minor-opcode: CARD16

This event is reported to clients selecting graphics-exposures in a graphics context and is generated when a graphics request that might produce GraphicsExposure events does not produce any. The drawable specifies the destination used for the graphics request.

The major and minor opcodes identify the graphics request used. For the core protocol, major-opcode is always CopyArea or CopyPlane, and the minor-opcode is always zero.

VisibilityNotify

window: WINDOW
state: {Unobscured, PartiallyObscured, FullyObscured}

This event is reported to clients selecting VisibilityChange on the window. In the following, the state of the window is calculated ignoring all of the window's subwindows. When a window changes state from partially or fully obscured or not viewable to viewable and completely unobscured, an event with Unobscured is generated. When a window changes state from a) viewable and completely unobscured or b) not viewable, to viewable and partially obscured, an event with PartiallyObscured is generated. When a window changes state from a) viewable and completely unobscured or b) viewable and partially obscured or c) not viewable, to viewable and fully obscured, an event with FullyObscured is generated.

VisibilityNotify events are never generated on InputOnly windows.

All VisibilityNotify events caused by a hierarchy change are generated after any hierarchy event (for example, UnmapNotify, MapNotify, ConfigureNotify, GravityNotify, CirculateNotify) caused by that change. Any VisibilityNotify event on a given window are generated before any Expose events on that window, but it is not required that all VisibilityNotify events on all windows be generated before all Expose events on all windows. The ordering of VisibilityNotify events with respect to FocusOut, EnterNotify, and LeaveNotify events is not constrained.

CreateNotify

parent, window: WINDOW
x, y: INT16
width, height, border-width: CARD16
override-redirect: BOOL

This event is reported to clients selecting SubstructureNotify on the parent and is generated when the window is created. The arguments are as in the CreateWindow request.

DestroyNotify

event, window: WINDOW

This event reported to clients selecting StructureNotify on the window and to clients selecting SubstructureNotify on the parent. It is generated when the window is destroyed. The event is the window on which the event was generated, and the window is the window that is destroyed.

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.

UnmapNotify

event, window: WINDOW
from-configure: BOOL

This event is reported to clients selecting StructureNotify on the window and to clients selecting SubstructureNotify on the parent. It is generated when the window changes state from mapped to unmapped. The event is the window on which the event was generated, and the window is the window that is unmapped. The from-configure flag is True if the event was generated as a result of the window's parent being resized when the window itself had a win-gravity of Unmap.

MapNotify

event, window: WINDOW
override-redirect: BOOL

This event is reported to clients selecting StructureNotify on the window and to clients selecting SubstructureNotify on the parent. It is generated when the window changes state from unmapped to mapped. The event is the window on which the event was generated, and the window is the window that is mapped. The override-redirect flag is from the window's attribute.

MapRequest

parent, window: WINDOW

This event is reported to the client selecting SubstructureRedirect on the parent and is generated when a MapWindow request is issued on an unmapped window with an override-redirect attribute of False.

ReparentNotify

event, window, parent: WINDOW
x, y: INT16
override-redirect: BOOL

This event is reported to clients selecting SubstructureNotify on either the old or the new parent and to clients selecting StructureNotify on the window. It is generated when the window is reparented. The event is the window on which the event was generated. The window is the window that has been re-rooted. The parent specifies the new parent. The x and y coordinates are relative to the new parent's origin and specify the position of the upper-left outer corner of the window. The override-redirect flag is from the window's attribute.

ConfigureNotify

event, window: WINDOW
x, y: INT16
width, height, border-width: CARD16
above-sibling: WINDOW or None
override-redirect: BOOL

This event is reported to clients selecting StructureNotify on the window and to clients selecting SubstructureNotify on the parent. It is generated when a ConfigureWindow request actually changes the state of the window. The event is the window on which the event was generated, and the window is the window that is changed. The x and y coordinates are relative to the new parent's origin and specify the position of the upper left outer corner of the window. The width and height specify the inside size, not including the border. If above-sibling is None, then the window is on the bottom of the stack with respect to siblings. Otherwise, the window is immediately on top of the specified sibling. The override-redirect flag is from the window's attribute.

GravityNotify

event, window: WINDOW
x, y: INT16

This event is reported to clients selecting SubstructureNotify on the parent and to clients selecting StructureNotify on the window. It is generated when a window is moved because of a change in size of the parent. The event is the window on which the event was generated, and the window is the window that is moved. The x and y coordinates are relative to the new parent's origin and specify the position of the upper left outer corner of the window.

ResizeRequest

window: WINDOW
width, height: CARD16

This event is reported to the client selecting ResizeRedirect on the window and is generated when a ConfigureWindow request by some other client on the window attempts to change the size of the window. The width and height are the inside size, not including the border.

ConfigureRequest

parent, window: WINDOW
x, y: INT16
width, height, border-width: CARD16
sibling: WINDOW or None
stack-mode: {Above, Below, TopIf, BottomIf, Opposite}
value-mask: BITMASK

This event is reported to the client selecting SubstructureRedirect on the parent and is generated when a ConfigureWindow request is issued on the window by some other client. The value-mask indicates which components were specified in the request. The value-mask and the corresponding values are reported as given in the request. The remaining values are filled in from the current geometry of the window, except in the case of sibling and stack-mode, which are reported as Above and None (respectively) if not given in the request.

CirculateNotify

event, window: WINDOW
place: {Top, Bottom}

This event is reported to clients selecting StructureNotify on the window and to clients selecting SubstructureNotify on the parent. It is generated when the window is actually restacked from a CirculateWindow request. The event is the window on which the event was generated, and the window is the window that is restacked. If place is Top, the window is now on top of all siblings. Otherwise, it is below all siblings.

CirculateRequest

parent, window: WINDOW
place: {Top, Bottom}

This event is reported to the client selecting SubstructureRedirect on the parent and is generated when a CirculateWindow request is issued on the parent and a window actually needs to be restacked. The window specifies the window to be restacked, and the place specifies what the new position in the stacking order should be.

PropertyNotify

window: WINDOW
atom: ATOM
state: {NewValue, Deleted}
time: TIMESTAMP

This event is reported to clients selecting PropertyChange on the window and is generated with state NewValue when a property of the window is changed using ChangeProperty or RotateProperties, even when adding zero-length data using ChangeProperty and when replacing all or part of a property with identical data using ChangeProperty or RotateProperties. It is generated with state Deleted when a property of the window is deleted using request DeleteProperty or GetProperty. The timestamp indicates the server time when the property was changed.

SelectionClear

owner: WINDOW
selection: ATOM
time: TIMESTAMP

This event is reported to the current owner of a selection and is generated when a new owner is being defined by means of SetSelectionOwner. The timestamp is the last-change time recorded for the selection. The owner argument is the window that was specified by the current owner in its SetSelectionOwner request.

SelectionRequest

owner: WINDOW
selection: ATOM
target: ATOM
property: ATOM or None
requestor: WINDOW
time: TIMESTAMP or CurrentTime

This event is reported to the owner of a selection and is generated when a client issues a ConvertSelection request. The owner argument is the window that was specified in the SetSelectionOwner request. The remaining arguments are as in the ConvertSelection request.

The owner should convert the selection based on the specified target type. If a property is specified, the owner should store the result as that property on the requestor window and then send a SelectionNotify event to the requestor using SendEvent with an empty event-mask (that is, the event should be sent to the creator of the requestor window). If None is specified as the property, the owner should choose a property name and store the result as that property on the requestor window, and then send a SelectionNotify giving that actual property name. If the selection cannot be converted as requested, the owner should send a SelectionNotify with the property set to None.

SelectionNotify

requestor: WINDOW
selection, *target*: ATOM
property: ATOM or None
time: TIMESTAMP or CurrentTime

This event is generated by the server in response to a ConvertSelection request when there is no owner for the selection. When there is an owner, it should be generated by the owner using SendEvent. The owner of a selection should send this event to a requestor either when a selection has been converted and stored as a property or when a selection conversion could not be performed (indicated with property None).

ColormapNotify

window: WINDOW
colormap: COLORMAP or None
new: BOOL
state: {Installed, Uninstalled}

This event is reported to clients selecting ColormapChange on the window. It is generated with value True for new when the colormap attribute of the window is changed and is generated with value False for new when the colormap of a whether the colormap is currently installed.

MappingNotify

request: {Modifier, Keyboard, Pointer}
first-keycode, *count*: CARD8

This event is sent to all clients, regardless. There is no mechanism to express disinterest in this event. The detail indicates the kind of change that occurred: Modifiers for a successful

SetModifierMapping, Keyboard for a successful ChangeKeyboardMapping, and Pointer for a successful SetPointerMapping. If the detail is Keyboard, then first-keycode and count indicate the range of altered keycodes.

ClientMessage

window: WINDOW
type: ATOM
format: {8, 16, 32}
data: LISTofINT8 or LISTofINT16 or LISTofINT32

This event is only generated by clients using SendEvent. The type specifies how the data is to be interpreted by the receiving client; the server places no interpretation on the type or the data. The format specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities, so that the server can correctly byte-swap, as necessary. The data always consists of either 20 8-bit values or 10 16-bit values or 5 32-bit values, although particular message types might not make use of all of these values.

13. Flow Control and Concurrency

Whenever the server is writing to a given connection, it is permissible for the server to stop reading from that connection (but if the writing would block it must continue to service other connections). The server is not required to buffer more than a single request per connection at one time. For a given connection to the server, a client can block while reading from the connection but should undertake to read (events and errors) when writing would block. Failure on the part of a client to obey this rule could result in a deadlocked connection, although deadlock is probably unlikely unless either the transport layer has very little buffering or the client attempts to send large numbers of requests without ever reading replies or checking for errors and events.

If a server is implemented with internal concurrency, the overall effect must be as if individual requests are executed to completion in some serial order, and that requests from a given connection are executed in delivery order (that is, the total execution order is a shuffle of the individual streams). The execution of a request includes validating all arguments, collecting all data for any reply, and generating (and queueing) all required events, but it does not include the actual transmission of the reply and the events. In addition, the effect of any other cause (for example, activation of a grab, pointer motion) that can generate multiple events must effectively generate (and queue) all required events indivisibly with respect to all other causes and requests.

Appendix A

KEYSYM Encoding

KEYSYM values are, for convenience, viewed as split into four bytes:

- Byte 1 (for the purposes of this document) is the most significant 5 bits (because of the 29-bit effective values)
- Byte 2 is the next most significant 8 bits
- Byte 3 is the next most significant 8 bits
- Byte 4 is the least significant 8 bits

The standard KEYSYMs all have the zero values for bytes 1 and 2, with byte 3 indicating a character code set and byte 4 indicating a particular character within that set:

Byte 3	Byte 4
0	Latin 1
1	Latin 2
2	Latin 3
3	Latin 4
4	Kana
5	Arabic
6	Cyrillic
7	Greek
8	Technical
9	Special
10	Publishing
11	APL
12	Hebrew
255	Keyboard

Each character set contains gaps where codes have been removed that were duplicates with codes in previous (that is, with lesser byte 3 value) character sets.

The 94 and 96 character code sets have been moved to occupy the right hand quadrant (decimal 129 - 256), so the ASCII subset has a unique encoding across byte 4 which corresponds to the ASCII character code. However, this cannot be guaranteed with future registrations and does not apply to all of the Keyboard set.

To the best of our knowledge, the Latin, Kana, Arabic, Cyrillic, Greek, Technical, APL, and Hebrew sets are from the appropriate ISO and/or ECMA international standards. There are no Technical, Special nor Publishing international standards, so these sets are based on Digital Equipment Corporation standards.

The ordering between the sets (byte 3) sets is essentially arbitrary. Although the national and international standards bodies are commencing deliberations regarding international two and four byte character sets, we do not know of any proposed layouts.

The order may be arbitrary, but it is important in dealing with duplicate coding. As far as possible, keysym codes are the same as the character code. In the Latin-1 to Latin-4 sets, all duplicate glyphs occupy the same position. However, duplicates between Greek and Technical do not occupy the same code position. Thus, applications wishing to use the technical character set must transform the keysym via an array.

There is a difference between European and US usage of the names Pilcrow, Paragraph, and Section:

US name	European name	code position in Latin-1
Section sign	Paragraph sign	10/07
Paragraph sign	Pilcrow sign	11/06

We have adopted the names used by both the ISO and ECMA standards. Thus, 11/06 is Pilcrow sign, and 10/07 is Paragraph sign, section sign. Note that this favors the European usage.

The Keyboard set is a miscellaneous collection of commonly occuring keys on keyboards. Within this set, the keypad symbols are generally duplicates of symbols found on keys on the "main" part of the keyboard but are distinguished here because they often have a distinguishable semantics associated with them.

Keyboards tend to be comparatively standard with respect to the alphanumeric keys, but they radically differ on the miscellaneous function keys. Many function keys are left over from early timesharing days or are designed for a specific application. Keyboard layouts from large manufacturers tend to have lots of keys for every conceivable purpose, whereas small workstation manufacturers often add keys that are solely for support of some of their unique functionality. There are two ways of thinking about how to define keysyms for such a world:

- The "Engraving" approach
- The "Common" approach

The Engraving approach is to create a keysym for every unique key engraving. This is effectively taking the union of all key engravings on all keyboards. For example, some keyboards label function keys across the top as F1 through Fn, others label them as PF1 through PFn. These would be different keys under the Engraving approach. Likewise, "Lock" would differ from "Shift Lock", which is different from the up-arrow symbol that has the effect of changing lower-case to capital. There are lots of other aliases such as "Del", "DEL", "Delete", "Remove", and so forth. The Engraving approach makes it easy to decide if a new entry should be added to the keysym set: if it does not exactly match an existing one, then a new one is created. One estimate is that there would be on the order of 300-500 Keyboard keysyms using this approach, not counting foreign translations and variations.

The Common approach tries to capture all of the keys present on an interesting number of keyboards, folding likely aliases into the same keysym. For example, "Del", "DEL", and "Delete" are all merged into a single keysym. Vendors would be expected to augment the keysym set (using the vendor-specific encoding space) to include all of their unique keys that were not included in the standard set. Each vendor decides which of its keys map into the standard keysyms, which presumably can be overridden by a user. It is more difficult to implement this approach, since judgement is required on when a sufficient set of keyboards implement an engraving to justify making it a keysym in the standard set and on which engravings should be merged into a single keysym. Under this scheme there are an estimated 100-150 keysyms.

While neither scheme is perfect or elegant, the Common approach has been selected because it makes it easier to write a portable application. Having the Delete functionality merged into a single keysym allows an application to implement a deletion function and expect reasonable bindings on a wide set of workstations. Under the Common approach, application writers are still free to look for and interpret vendor-specific keysyms, but, because they are in the extended set, the application developer is more conscious that they are writing the application in a non-portable fashion.

In the listings below, Code Pos is a representation of byte4 of the KEYSYM value, expressed as most-significant/least-significant 4-bit values. The Code Pos numbers are for reference only and do not affect the KEYSYM value. In all cases, the KEYSYM value is

byte3 * 256 + byte4

Byte 3	Byte 4	Code Pos	Name	Set
000	032	02/00	SPACE	Latin-1
000	033	02/01	EXCLAMATION POINT	Latin-1
000	034	02/02	QUOTATION MARK	Latin-1
000	035	02/03	NUMBER SIGN	Latin-1
000	036	02/04	DOLLAR SIGN	Latin-1
000	037	02/05	PERCENT SIGN	Latin-1
000	038	02/06	AMPERSAND	Latin-1
000	039	02/07	APOSTROPHE	Latin-1
000	040	02/08	LEFT PARENTHESIS	Latin-1
000	041	02/09	RIGHT PARENTHESIS	Latin-1
000	042	02/10	ASTERISK	Latin-1
000	043	02/11	PLUS SIGN	Latin-1
000	044	02/12	COMMA	Latin-1
000	045	02/13	HYPHEN, MINUS SIGN	Latin-1
000	046	02/14	FULL STOP	Latin-1
000	047	02/15	SOLIDUS	Latin-1
000	048	03/00	DIGIT ZERO	Latin-1
000	049	03/01	DIGIT ONE	Latin-1
000	050	03/02	DIGIT TWO	Latin-1
000	051	03/03	DIGIT THREE	Latin-1
000	052	03/04	DIGIT FOUR	Latin-1
000	053	03/05	DIGIT FIVE	Latin-1
000	054	03/06	DIGIT SIX	Latin-1
000	055	03/07	DIGIT SEVEN	Latin-1

Byte 3	Byte 4	Code Pos	Name	Set
000	056	03/08	DIGIT EIGHT	Latin-1
000	057	03/09	DIGIT NINE	Latin-1
000	058	03/10	COLON	Latin-1
000	059	03/11	SEMICOLON	Latin-1
000	060	03/12	LESS THAN SIGN	Latin-1
000	061	03/13	EQUALS SIGN	Latin-1
000	062	03/14	GREATER THAN SIGN	Latin-1
000	063	03/15	QUESTION MARK	Latin-1
000	064	04/00	COMMERCIAL AT	Latin-1
000	065	04/01	LATIN CAPITAL LETTER A	Latin-1
000	066	04/02	LATIN CAPITAL LETTER B	Latin-1
000	067	04/03	LATIN CAPITAL LETTER C	Latin-1
000	068	04/04	LATIN CAPITAL LETTER D	Latin-1
000	069	04/05	LATIN CAPITAL LETTER E	Latin-1
000	070	04/06	LATIN CAPITAL LETTER F	Latin-1
000	071	04/07	LATIN CAPITAL LETTER G	Latin-1
000	072	04/08	LATIN CAPITAL LETTER H	Latin-1
000	073	04/09	LATIN CAPITAL LETTER I	Latin-1
000	074	04/10	LATIN CAPITAL LETTER J	Latin-1
000	075	04/11	LATIN CAPITAL LETTER K	Latin-1
000	076	04/12	LATIN CAPITAL LETTER L	Latin-1
000	077	04/13	LATIN CAPITAL LETTER M	Latin-1
000	078	04/14	LATIN CAPITAL LETTER N	Latin-1
000	079	04/15	LATIN CAPITAL LETTER O	Latin-1
000	080	05/00	LATIN CAPITAL LETTER P	Latin-1
000	081	05/01	LATIN CAPITAL LETTER Q	Latin-1
000	082	05/02	LATIN CAPITAL LETTER R	Latin-1
000	083	05/03	LATIN CAPITAL LETTER S	Latin-1
000	084	05/04	LATIN CAPITAL LETTER T	Latin-1
000	085	05/05	LATIN CAPITAL LETTER U	Latin-1
000	086	05/06	LATIN CAPITAL LETTER V	Latin-1
000	087	05/07	LATIN CAPITAL LETTER W	Latin-1
000	088	05/08	LATIN CAPITAL LETTER X	Latin-1
000	089	05/09	LATIN CAPITAL LETTER Y	Latin-1
000	090	05/10	LATIN CAPITAL LETTER Z	Latin-1
000	091	05/11	LEFT SQUARE BRACKET	Latin-1
000	092	05/12	REVERSE SOLIDUS	Latin-1
000	093	05/13	RIGHT SQUARE BRACKET	Latin-1
000	094	05/14	CIRCUMFLEX ACCENT	Latin-1
000	095	05/15	LOW LINE	Latin-1
000	096	06/00	GRAVE ACCENT	Latin-1
000	097	06/01	LATIN SMALL LETTER a	Latin-1
000	098	06/02	LATIN SMALL LETTER b	Latin-1
000	099	06/03	LATIN SMALL LETTER c	Latin-1
000	100	06/04	LATIN SMALL LETTER d	Latin-1
000	101	06/05	LATIN SMALL LETTER e	Latin-1
000	102	06/06	LATIN SMALL LETTER f	Latin-1
000	103	06/07	LATIN SMALL LETTER g	Latin-1
000	104	06/08	LATIN SMALL LETTER h	Latin-1
000	105	06/09	LATIN SMALL LETTER i	Latin-1
000	106	06/10	LATIN SMALL LETTER j	Latin-1
000	107	06/11	LATIN SMALL LETTER k	Latin-1
000	108	06/12	LATIN SMALL LETTER l	Latin-1
000	109	06/13	LATIN SMALL LETTER m	Latin-1
000	110	06/14	LATIN SMALL LETTER n	Latin-1
000	111	06/15	LATIN SMALL LETTER o	Latin-1
000	112	07/00	LATIN SMALL LETTER p	Latin-1
000	113	07/01	LATIN SMALL LETTER q	Latin-1
000	114	07/02	LATIN SMALL LETTER r	Latin-1
000	115	07/03	LATIN SMALL LETTER s	Latin-1
000	116	07/04	LATIN SMALL LETTER t	Latin-1

Byte 3	Byte 4	Code Pos	Name	Set
000	117	07/05	LATIN SMALL LETTER u	Latin-1
000	118	07/06	LATIN SMALL LETTER v	Latin-1
000	119	07/07	LATIN SMALL LETTER w	Latin-1
000	120	07/08	LATIN SMALL LETTER x	Latin-1
000	121	07/09	LATIN SMALL LETTER y	Latin-1
000	122	07/10	LATIN SMALL LETTER z	Latin-1
000	123	07/11	LEFT CURLY BRACKET	Latin-1
000	124	07/12	VERTICAL LINE	Latin-1
000	125	07/13	RIGHT CURLY BRACKET	Latin-1
000	126	07/14	TILDE	Latin-1
000	160	10/00	NO-BREAK SPACE	Latin-1
000	161	10/01	INVERTED EXCLAMATION MARK	Latin-1
000	162	10/02	CENT SIGN	Latin-1
000	163	10/03	POUND SIGN	Latin-1
000	164	10/04	CURRENCY SIGN	Latin-1
000	165	10/05	YEN SIGN	Latin-1
000	166	10/06	BROKEN VERTICAL BAR	Latin-1
000	167	10/07	PARAGRAPH SIGN, SECTION SIGN	Latin-1
000	168	10/08	DIAERESIS	Latin-1
000	169	10/09	COPYRIGHT SIGN	Latin-1
000	170	10/10	FEMININE ORDINAL INDICATOR	Latin-1
000	171	10/11	LEFT ANGLE QUOTATION MARK	Latin-1
000	172	10/12	NOT SIGN	Latin-1
000	174	10/14	REGISTERED TRADE MARK SIGN	Latin-1
000	175	10/15	MACRON	Latin-1
000	176	11/00	DEGREE SIGN, RING ABOVE	Latin-1
000	177	11/01	PLUS-MINUS SIGN	Latin-1
000	178	11/02	SUPERSCRIPT TWO	Latin-1
000	179	11/03	SUPERSCRIPT THREE	Latin-1
000	180	11/04	ACUTE ACCENT	Latin-1
000	181	11/05	MICRO SIGN	Latin-1
000	182	11/06	PILCROW SIGN	Latin-1
000	183	11/07	MIDDLE DOT	Latin-1
000	184	11/08	CEDILLA	Latin-1
000	185	11/09	SUPERSCRIPT ONE	Latin-1
000	186	11/10	MASCULINE ORDINAL INDICATOR	Latin-1
000	187	11/11	RIGHT ANGLE QUOTATION MARK	Latin-1
000	188	11/12	VULGAR FRACTION ONE QUARTER	Latin-1
000	189	11/13	VULGAR FRACTION ONE HALF	Latin-1
000	190	11/14	VULGAR FRACTION THREE QUARTERS	Latin-1
000	191	11/15	INVERTED QUESTION MARK	Latin-1
000	192	12/00	LATIN CAPITAL LETTER A WITH GRAVE ACCENT	Latin-1
000	193	12/01	LATIN CAPITAL LETTER A WITH ACUTE ACCENT	Latin-1
000	194	12/02	LATIN CAPITAL LETTER A WITH CIRCUMFLEX ACCENT	Latin-1
000	195	12/03	LATIN CAPITAL LETTER A WITH TILDE	Latin-1
000	196	12/04	LATIN CAPITAL LETTER A WITH DIAERESIS	Latin-1
000	197	12/05	LATIN CAPITAL LETTER A WITH RING ABOVE	Latin-1
000	198	12/06	LATIN CAPITAL DIPHTHONG AE	Latin-1
000	199	12/07	LATIN CAPITAL LETTER C WITH CEDILLA	Latin-1
000	200	12/08	LATIN CAPITAL LETTER E WITH GRAVE ACCENT	Latin-1
000	201	12/09	LATIN CAPITAL LETTER E WITH ACUTE ACCENT	Latin-1
000	202	12/10	LATIN CAPITAL LETTER E WITH CIRCUMFLEX ACCENT	Latin-1
000	203	12/11	LATIN CAPITAL LETTER E WITH DIAERESIS	Latin-1
000	204	12/12	LATIN CAPITAL LETTER I WITH GRAVE ACCENT	Latin-1
000	205	12/13	LATIN CAPITAL LETTER I WITH ACUTE ACCENT	Latin-1
000	206	12/14	LATIN CAPITAL LETTER I WITH CIRCUMFLEX ACCENT	Latin-1
000	207	12/15	LATIN CAPITAL LETTER I WITH DIAERESIS	Latin-1
000	208	13/00	ICELANDIC CAPITAL LETTER ETH	Latin-1
000	209	13/01	LATIN CAPITAL LETTER N WITH TILDE	Latin-1
000	210	13/02	LATIN CAPITAL LETTER O WITH GRAVE ACCENT	Latin-1
000	211	13/03	LATIN CAPITAL LETTER O WITH ACUTE ACCENT	Latin-1

Byte 3	Byte 4	Code Pos	Name	Set
000	212	13/04	LATIN CAPITAL LETTER O WITH CIRCUMFLEX ACCENT	Latin-1
000	213	13/05	LATIN CAPITAL LETTER O WITH TILDE	Latin-1
000	214	13/06	LATIN CAPITAL LETTER O WITH DIAERESIS	Latin-1
000	215	13/07	MULTIPLICATION SIGN	Latin-1
000	216	13/08	LATIN CAPITAL LETTER O WITH OBLIQUE STROKE	Latin-1
000	217	13/09	LATIN CAPITAL LETTER U WITH GRAVE ACCENT	Latin-1
000	218	13/10	LATIN CAPITAL LETTER U WITH ACUTE ACCENT	Latin-1
000	219	13/11	LATIN CAPITAL LETTER U WITH CIRCUMFLEX ACCENT	Latin-1
000	220	13/12	LATIN CAPITAL LETTER U WITH DIAERESIS	Latin-1
000	221	13/13	LATIN CAPITAL LETTER Y WITH ACUTE ACCENT	Latin-1
000	222	13/14	ICELANDIC CAPITAL LETTER THORN	Latin-1
000	223	13/15	GERMAN SMALL LETTER SHARP s	Latin-1
000	224	14/00	LATIN SMALL LETTER a WITH GRAVE ACCENT	Latin-1
000	225	14/01	LATIN SMALL LETTER a WITH ACUTE ACCENT	Latin-1
000	226	14/02	LATIN SMALL LETTER a WITH CIRCUMFLEX ACCENT	Latin-1
000	227	14/03	LATIN SMALL LETTER a WITH TILDE	Latin-1
000	228	14/04	LATIN SMALL LETTER a WITH DIAERESIS	Latin-1
000	229	14/05	LATIN SMALL LETTER a WITH RING ABOVE	Latin-1
000	230	14/06	LATIN SMALL DIPHTHONG ae	Latin-1
000	231	14/07	LATIN SMALL LETTER c WITH CEDILLA	Latin-1
000	232	14/08	LATIN SMALL LETTER e WITH GRAVE ACCENT	Latin-1
000	233	14/09	LATIN SMALL LETTER e WITH ACUTE ACCENT	Latin-1
000	234	14/10	LATIN SMALL LETTER e WITH CIRCUMFLEX ACCENT	Latin-1
000	235	14/11	LATIN SMALL LETTER e WITH DIAERESIS	Latin-1
000	236	14/12	LATIN SMALL LETTER i WITH GRAVE ACCENT	Latin-1
000	237	14/13	LATIN SMALL LETTER i WITH ACUTE ACCENT	Latin-1
000	238	14/14	LATIN SMALL LETTER i WITH CIRCUMFLEX ACCENT	Latin-1
000	239	14/15	LATIN SMALL LETTER i WITH DIAERESIS	Latin-1
000	240	15/00	ICELANDIC SMALL LETTER ETH	Latin-1
000	241	15/01	LATIN SMALL LETTER n WITH TILDE	Latin-1
000	242	15/02	LATIN SMALL LETTER o WITH GRAVE ACCENT	Latin-1
000	243	15/03	LATIN SMALL LETTER o WITH ACUTE ACCENT	Latin-1
000	244	15/04	LATIN SMALL LETTER o WITH CIRCUMFLEX ACCENT	Latin-1
000	245	15/05	LATIN SMALL LETTER o WITH TILDE	Latin-1
000	246	15/06	LATIN SMALL LETTER o WITH DIAERESIS	Latin-1
000	247	15/07	DIVISION SIGN	Latin-1
000	248	15/08	LATIN SMALL LETTER o WITH OBLIQUE STROKE	Latin-1
000	249	15/09	LATIN SMALL LETTER u WITH GRAVE ACCENT	Latin-1
000	250	15/10	LATIN SMALL LETTER u WITH ACUTE ACCENT	Latin-1
000	251	15/11	LATIN SMALL LETTER u WITH CIRCUMFLEX ACCENT	Latin-1
000	252	15/12	LATIN SMALL LETTER u WITH DIAERESIS	Latin-1
000	253	15/13	LATIN SMALL LETTER y WITH ACUTE ACCENT	Latin-1
000	254	15/14	ICELANDIC SMALL LETTER THORN	Latin-1
000	255	15/15	LATIN SMALL LETTER y WITH DIAERESIS	Latin-1
001	161	10/01	LATIN CAPITAL LETTER A WITH OGONEK	Latin-2
001	162	10/02	BREVE	Latin-2
001	163	10/03	LATIN CAPITAL LETTER L WITH STROKE	Latin-2
001	165	10/05	LATIN CAPITAL LETTER L WITH CARON	Latin-2
001	166	10/06	LATIN CAPITAL LETTER S WITH ACUTE ACCENT	Latin-2
001	169	10/09	LATIN CAPITAL LETTER S WITH CARON	Latin-2
001	170	10/10	LATIN CAPITAL LETTER S WITH CEDILLA	Latin-2
001	171	10/11	LATIN CAPITAL LETTER T WITH CARON	Latin-2
001	172	10/12	LATIN CAPITAL LETTER Z WITH ACUTE ACCENT	Latin-2
001	174	10/14	LATIN CAPITAL LETTER Z WITH CARON	Latin-2
001	175	10/15	LATIN CAPITAL LETTER Z WITH DOT ABOVE	Latin-2
001	177	11/01	LATIN SMALL LETTER a WITH OGONEK	Latin-2
001	178	11/02	OGONEK	Latin-2
001	179	11/03	LATIN SMALL LETTER l WITH STROKE	Latin-2
001	181	11/05	LATIN SMALL LETTER l WITH CARON	Latin-2

Byte 3	Byte 4	Code Pos	Name	Set
001	182	11/06	LATIN SMALL LETTER s WITH ACUTE ACCENT	Latin-2
001	183	11/07	CARON	Latin-2
001	185	11/09	LATIN SMALL LETTER s WITH CARON	Latin-2
001	186	11/10	LATIN SMALL LETTER s WITH CEDILLA	Latin-2
001	187	11/11	LATIN SMALL LETTER t WITH CARON	Latin-2
001	188	11/12	LATIN SMALL LETTER z WITH ACUTE ACCENT	Latin-2
001	189	11/13	DOUBLE ACUTE ACCENT	Latin-2
001	190	11/14	LATIN SMALL LETTER z WITH CARON	Latin-2
001	191	11/15	LATIN SMALL LETTER z WITH DOT ABOVE	Latin-2
001	192	12/00	LATIN CAPITAL LETTER R WITH ACUTE ACCENT	Latin-2
001	195	12/03	LATIN CAPITAL LETTER A WITH BREVE	Latin-2
001	197	12/05	LATIN CAPITAL LETTER L WITH ACUTE ACCENT	Latin-2
001	198	12/06	LATIN CAPITAL LETTER C WITH ACUTE ACCENT	Latin-2
001	200	12/08	LATIN CAPITAL LETTER C WITH CARON	Latin-2
001	202	12/10	LATIN CAPITAL LETTER E WITH OGONEK	Latin-2
001	204	12/12	LATIN CAPITAL LETTER E WITH CARON	Latin-2
001	207	12/15	LATIN CAPITAL LETTER D WITH CARON	Latin-2
001	208	13/00	LATIN CAPITAL LETTER D WITH STROKE	Latin-2
001	209	13/01	LATIN CAPITAL LETTER N WITH ACUTE ACCENT	Latin-2
001	210	13/02	LATIN CAPITAL LETTER N WITH CARON	Latin-2
001	213	13/05	LATIN CAPITAL LETTER O WITH DOUBLE ACUTE ACCENT	Latin-2
001	216	13/08	LATIN CAPITAL LETTER R WITH CARON	Latin-2
001	217	13/09	LATIN CAPITAL LETTER U WITH RING ABOVE	Latin-2
001	219	13/11	LATIN CAPITAL LETTER U WITH DOUBLE ACUTE ACCENT	Latin-2
001	222	13/14	LATIN CAPITAL LETTER T WITH CEDILLA	Latin-2
001	224	14/00	LATIN SMALL LETTER r WITH ACUTE ACCENT	Latin-2
001	227	14/03	LATIN SMALL LETTER a WITH BREVE	Latin-2
001	229	14/05	LATIN SMALL LETTER l WITH ACUTE ACCENT	Latin-2
001	230	14/06	LATIN SMALL LETTER c WITH ACUTE ACCENT	Latin-2
001	232	14/08	LATIN SMALL LETTER c WITH CARON	Latin-2
001	234	14/10	LATIN SMALL LETTER e WITH OGONEK	Latin-2
001	236	14/12	LATIN SMALL LETTER e WITH CARON	Latin-2
001	239	14/15	LATIN SMALL LETTER d WITH CARON	Latin-2
001	240	15/00	LATIN SMALL LETTER d WITH STROKE	Latin-2
001	241	15/01	LATIN SMALL LETTER n WITH ACUTE ACCENT	Latin-2
001	242	15/02	LATIN SMALL LETTER n WITH CARON	Latin-2
001	245	15/05	LATIN SMALL LETTER o WITH DOUBLE ACUTE ACCENT	Latin-2
001	248	15/08	LATIN SMALL LETTER r WITH CARON	Latin-2
001	249	15/09	LATIN SMALL LETTER u WITH RING ABOVE	Latin-2
001	251	15/11	LATIN SMALL LETTER u WITH DOUBLE ACUTE ACCENT	Latin-2
001	254	15/14	LATIN SMALL LETTER t WITH CEDILLA	Latin-2
001	255	15/15	DOT ABOVE	Latin-2
002	161	10/01	LATIN CAPITAL LETTER H WITH STROKE	Latin-3
002	166	10/06	LATIN CAPITAL LETTER H WITH CIRCUMFLEX ACCENT	Latin-3
002	169	10/09	LATIN CAPITAL LETTER I WITH DOT ABOVE	Latin-3
002	171	10/11	LATIN CAPITAL LETTER G WITH BREVE	Latin-3
002	172	10/12	LATIN CAPITAL LETTER J WITH CIRCUMFLEX ACCENT	Latin-3
002	177	11/01	LATIN SMALL LETTER h WITH STROKE	Latin-3
002	182	11/06	LATIN SMALL LETTER h WITH CIRCUMFLEX ACCENT	Latin-3
002	185	11/09	SMALL DOTLESS LETTER i	Latin-3
002	187	11/11	LATIN SMALL LETTER g WITH BREVE	Latin-3
002	188	11/12	LATIN SMALL LETTER j WITH CIRCUMFLEX ACCENT	Latin-3
002	197	12/05	LATIN CAPITAL LETTER C WITH DOT ABOVE	Latin-3
002	198	12/06	LATIN CAPITAL LETTER C WITH CIRCUMFLEX ACCENT	Latin-3
002	213	13/05	LATIN CAPITAL LETTER G WITH DOT ABOVE	Latin-3
002	216	13/08	LATIN CAPITAL LETTER G WITH CIRCUMFLEX ACCENT	Latin-3
002	221	13/13	LATIN CAPITAL LETTER U WITH BREVE	Latin-3
002	222	13/14	LATIN CAPITAL LETTER S WITH CIRCUMFLEX ACCENT	Latin-3
002	229	14/05	LATIN SMALL LETTER c WITH DOT ABOVE	Latin-3

Byte 3	Byte 4	Code Pos	Name	Set
002	230	14/06	LATIN SMALL LETTER c WITH CIRCUMFLEX ACCENT	Latin-3
002	245	15/05	LATIN SMALL LETTER g WITH DOT ABOVE	Latin-3
002	248	15/08	LATIN SMALL LETTER g WITH CIRCUMFLEX ACCENT	Latin-3
002	253	15/13	LATIN SMALL LETTER u WITH BREVE	Latin-3
002	254	15/14	LATIN SMALL LETTER s WITH CIRCUMFLEX ACCENT	Latin-3
003	162	10/02	LATIN SMALL LETTER KAPPA	Latin-4
003	163	10/03	LATIN CAPITAL LETTER R WITH CEDILLA	Latin-4
003	165	10/05	LATIN CAPITAL LETTER I WITH TILDE	Latin-4
003	166	10/06	LATIN CAPITAL LETTER L WITH CEDILLA	Latin-4
003	170	10/10	LATIN CAPITAL LETTER E WITH MACRON	Latin-4
003	171	10/11	LATIN CAPITAL LETTER G WITH CEDILLA	Latin-4
003	172	10/12	LATIN CAPITAL LETTER T WITH OBLIQUE STROKE	Latin-4
003	179	11/03	LATIN SMALL LETTER r WITH CEDILLA	Latin-4
003	181	11/05	LATIN SMALL LETTER i WITH TILDE	Latin-4
003	182	11/06	LATIN SMALL LETTER l WITH CEDILLA	Latin-4
003	186	11/10	LATIN SMALL LETTER e WITH MACRON	Latin-4
003	187	11/11	LATIN SMALL LETTER g WITH ACUTE ACCENT	Latin-4
003	188	11/12	LATIN SMALL LETTER t WITH OBLIQUE STROKE	Latin-4
003	189	11/13	LAPPISH CAPITAL LETTER ENG	Latin-4
003	191	11/15	LAPPISH SMALL LETTER ENG	Latin-4
003	192	12/00	LATIN CAPITAL LETTER A WITH MACRON	Latin-4
003	199	12/07	LATIN CAPITAL LETTER I WITH OGONEK	Latin-4
003	204	12/12	LATIN CAPITAL LETTER E WITH DOT ABOVE	Latin-4
003	207	12/15	LATIN CAPITAL LETTER I WITH MACRON	Latin-4
003	209	13/01	LATIN CAPITAL LETTER N WITH CEDILLA	Latin-4
003	210	13/02	LATIN CAPITAL LETTER O WITH MACRON	Latin-4
003	211	13/03	LATIN CAPITAL LETTER K WITH CEDILLA	Latin-4
003	217	13/09	LATIN CAPITAL LETTER U WITH OGONEK	Latin-4
003	221	13/13	LATIN CAPITAL LETTER U WITH TILDE	Latin-4
003	222	13/14	LATIN CAPITAL LETTER U WITH MACRON	Latin-4
003	224	14/00	LATIN SMALL LETTER a WITH MACRON	Latin-4
003	231	14/07	LATIN SMALL LETTER i WITH OGONEK	Latin-4
003	236	14/12	LATIN SMALL LETTER e WITH DOT ABOVE	Latin-4
003	239	14/15	LATIN SMALL LETTER i WITH MACRON	Latin-4
003	241	15/01	LATIN SMALL LETTER n WITH CEDILLA	Latin-4
003	242	15/02	LATIN SMALL LETTER o WITH MACRON	Latin-4
003	243	15/03	LATIN SMALL LETTER k WITH CEDILLA	Latin-4
003	249	15/09	LATIN SMALL LETTER u WITH OGONEK	Latin-4
003	253	15/13	LATIN SMALL LETTER u WITH TILDE	Latin-4
003	254	15/14	LATIN SMALL LETTER u WITH MACRON	Latin-4
004	126	07/14	OVERLINE	Kana
004	161	10/01	KANA FULL STOP	Kana
004	162	10/02	KANA OPENING BRACKET	Kana
004	163	10/03	KANA CLOSING BRACKET	Kana
004	164	10/04	KANA COMMA	Kana
004	165	10/05	KANA MIDDLE DOT	Kana
004	166	10/06	KANA LETTER WO	Kana
004	167	10/07	KANA LETTER SMALL A	Kana
004	168	10/08	KANA LETTER SMALL I	Kana
004	169	10/09	KANA LETTER SMALL U	Kana
004	170	10/10	KANA LETTER SMALL E	Kana
004	171	10/11	KANA LETTER SMALL O	Kana
004	172	10/12	KANA LETTER SMALL YA	Kana
004	173	10/13	KANA LETTER SMALL YU	Kana
004	174	10/14	KANA LETTER SMALL YO	Kana
004	175	10/15	KANA LETTER SMALL TU	Kana
004	176	11/00	PROLONGED SOUND SYMBOL	Kana

Byte 3	Byte 4	Code Pos	Name	Set
004	177	11/01	KANA LETTER A	Kana
004	178	11/02	KANA LETTER I	Kana
004	179	11/03	KANA LETTER U	Kana
004	180	11/04	KANA LETTER E	Kana
004	181	11/05	KANA LETTER O	Kana
004	182	11/06	KANA LETTER KA	Kana
004	183	11/07	KANA LETTER KI	Kana
004	184	11/08	KANA LETTER KU	Kana
004	185	11/09	KANA LETTER KE	Kana
004	186	11/10	KANA LETTER KO	Kana
004	187	11/11	KANA LETTER SA	Kana
004	188	11/12	KANA LETTER SHI	Kana
004	189	11/13	KANA LETTER SU	Kana
004	190	11/14	KANA LETTER SE	Kana
004	191	11/15	KANA LETTER SO	Kana
004	192	12/00	KANA LETTER TA	Kana
004	193	12/01	KANA LETTER TI	Kana
004	194	12/02	KANA LETTER TU	Kana
004	195	12/03	KANA LETTER TE	Kana
004	196	12/04	KANA LETTER TO	Kana
004	197	12/05	KANA LETTER NA	Kana
004	198	12/06	KANA LETTER NI	Kana
004	199	12/07	KANA LETTER NU	Kana
004	200	12/08	KANA LETTER NE	Kana
004	201	12/09	KANA LETTER NO	Kana
004	202	12/10	KANA LETTER HA	Kana
004	203	12/11	KANA LETTER HI	Kana
004	204	12/12	KANA LETTER HU	Kana
004	205	12/13	KANA LETTER HE	Kana
004	206	12/14	KANA LETTER HO	Kana
004	207	12/15	KANA LETTER MA	Kana
004	208	13/00	KANA LETTER MI	Kana
004	209	13/01	KANA LETTER MU	Kana
004	210	13/02	KANA LETTER ME	Kana
004	211	13/03	KANA LETTER MO	Kana
004	212	13/04	KANA LETTER YA	Kana
004	213	13/05	KANA LETTER YU	Kana
004	214	13/06	KANA LETTER YO	Kana
004	215	13/07	KANA LETTER RA	Kana
004	216	13/08	KANA LETTER RI	Kana
004	217	13/09	KANA LETTER RU	Kana
004	218	13/10	KANA LETTER RE	Kana
004	219	13/11	KANA LETTER RO	Kana
004	220	13/12	KANA LETTER WA	Kana
004	221	13/13	KANA LETTER N	Kana
004	222	13/14	VOICED SOUND SYMBOL	Kana
004	223	13/15	SEMI-VOICED SOUND SYMBOL	Kana
005	172	10/12	ARABIC COMMA	Arabic
005	187	11/11	ARABIC SEMI-COLON	Arabic
005	191	11/15	ARABIC QUESTION MARK	Arabic
005	193	12/01	ARABIC LETTER HAMZA	Arabic
005	194	12/02	ARABIC LETTER MADDA ON ALEF	Arabic
005	195	12/03	ARABIC LETTER HAMZA ON ALEF	Arabic
005	196	12/04	ARABIC LETTER HAMZA ON WAW	Arabic
005	197	12/05	ARABIC LETTER HAMZA UNDER ALEF	Arabic
005	198	12/06	ARABIC LETTER HAMZA ON YEH	Arabic
005	199	12/07	ARABIC LETTER ALEF	Arabic
005	200	12/08	ARABIC LETTER BEH	Arabic
005	201	12/09	ARABIC LETTER TEH MARBUTA	Arabic

Byte 3	Byte 4	Code Pos	Name	Set
005	202	12/10	ARABIC LETTER TEH	Arabic
005	203	12/11	ARABIC LETTER THEH	Arabic
005	204	12/12	ARABIC LETTER JEEM	Arabic
005	205	12/13	ARABIC LETTER HAH	Arabic
005	206	12/14	ARABIC LETTER KHAH	Arabic
005	207	12/15	ARABIC LETTER DAL	Arabic
005	208	13/00	ARABIC LETTER THAL	Arabic
005	209	13/01	ARABIC LETTER RA	Arabic
005	210	13/02	ARABIC LETTER ZAIN	Arabic
005	211	13/03	ARABIC LETTER SEEN	Arabic
005	212	13/04	ARABIC LETTER SHEEN	Arabic
005	213	13/05	ARABIC LETTER SAD	Arabic
005	214	13/06	ARABIC LETTER DAD	Arabic
005	215	13/07	ARABIC LETTER TAH	Arabic
005	216	13/08	ARABIC LETTER ZAH	Arabic
005	217	13/09	ARABIC LETTER AIN	Arabic
005	218	13/10	ARABIC LETTER GHAIN	Arabic
005	224	14/00	ARABIC LETTER TATWEEL	Arabic
005	225	14/01	ARABIC LETTER FEH	Arabic
005	226	14/02	ARABIC LETTER QAF	Arabic
005	227	14/03	ARABIC LETTER KAF	Arabic
005	228	14/04	ARABIC LETTER LAM	Arabic
005	229	14/05	ARABIC LETTER MEEM	Arabic
005	230	14/06	ARABIC LETTER NOON	Arabic
005	231	14/07	ARABIC LETTER HEH	Arabic
005	232	14/08	ARABIC LETTER WAW	Arabic
005	233	14/09	ARABIC LETTER ALEF MAKSURA	Arabic
005	234	14/10	ARABIC LETTER YEH	Arabic
005	235	14/11	ARABIC LETTER FATHATAN	Arabic
005	236	14/12	ARABIC LETTER DAMMATAN	Arabic
005	237	14/13	ARABIC LETTER KASRATAN	Arabic
005	238	14/14	ARABIC LETTER FATHA	Arabic
005	239	14/15	ARABIC LETTER DAMMA	Arabic
005	240	15/00	ARABIC LETTER KASRA	Arabic
005	241	15/01	ARABIC LETTER SHADDA	Arabic
005	242	15/02	ARABIC LETTER SUKUN	Arabic
006	161	10/01	SERBIAN SMALL LETTER DJE	Cyrillic
006	162	10/02	MACEDONIA SMALL LETTER GJE	Cyrillic
006	163	10/03	CYRILLIC SMALL LETTER IO	Cyrillic
006	164	10/04	UKRANIAN SMALL LETTER JE	Cyrillic
006	165	10/05	MACEDONIA SMALL LETTER DSE	Cyrillic
006	166	10/06	UKRANIAN SMALL LETTER I	Cyrillic
006	167	10/07	UKRANIAN SMALL LETTER YI	Cyrillic
006	168	10/08	SERBIAN SMALL LETTER JE	Cyrillic
006	169	10/09	SERBIAN SMALL LETTER LJE	Cyrillic
006	170	10/10	SERBIAN SMALL LETTER NJE	Cyrillic
006	171	10/11	SERBIAN SMALL LETTER TSHE	Cyrillic
006	172	10/12	MACEDONIA SMALL LETTER KJE	Cyrillic
006	174	10/14	BYELORUSSIAN SMALL LETTER SHORT U	Cyrillic
006	175	10/15	SERBIAN SMALL LETTER DZE	Cyrillic
006	176	11/00	NUMERO SIGN	Cyrillic
006	177	11/01	SERBIAN CAPITAL LETTER DJE	Cyrillic
006	178	11/02	MACEDONIA CAPITAL LETTER GJE	Cyrillic
006	179	11/03	CYRILLIC CAPITAL LETTER IO	Cyrillic
006	180	11/04	UKRANIAN CAPITAL LETTER JE	Cyrillic
006	181	11/05	MACEDONIA CAPITAL LETTER DSE	Cyrillic
006	182	11/06	UKRANIAN CAPITAL LETTER I	Cyrillic
006	183	11/07	UKRANIAN CAPITAL LETTER YI	Cyrillic
006	184	11/08	SERBIAN CAPITAL LETTER JE	Cyrillic

Byte 3	Byte 4	Code Pos	Name	Set
006	185	11/09	SERBIAN CAPITAL LETTER LJE	Cyrillic
006	186	11/10	SERBIAN CAPITAL LETTER NJE	Cyrillic
006	187	11/11	SERBIAN CAPITAL LETTER TSHE	Cyrillic
006	188	11/12	MACEDONIA CAPITAL LETTER KJE	Cyrillic
006	190	11/14	BYELORUSSIAN CAPITAL LETTER SHORT U	Cyrillic
006	191	11/15	SERBIAN CAPITAL LETTER DZE	Cyrillic
006	192	12/00	CYRILLIC SMALL LETTER YU	Cyrillic
006	193	12/01	CYRILLIC SMALL LETTER A	Cyrillic
006	194	12/02	CYRILLIC SMALL LETTER BE	Cyrillic
006	195	12/03	CYRILLIC SMALL LETTER TSE	Cyrillic
006	196	12/04	CYRILLIC SMALL LETTER DE	Cyrillic
006	197	12/05	CYRILLIC SMALL LETTER IE	Cyrillic
006	198	12/06	CYRILLIC SMALL LETTER EF	Cyrillic
006	199	12/07	CYRILLIC SMALL LETTER GHE	Cyrillic
006	200	12/08	CYRILLIC SMALL LETTER HA	Cyrillic
006	201	12/09	CYRILLIC SMALL LETTER I	Cyrillic
006	202	12/10	CYRILLIC SMALL LETTER SHORT I	Cyrillic
006	203	12/11	CYRILLIC SMALL LETTER KA	Cyrillic
006	204	12/12	CYRILLIC SMALL LETTER EL	Cyrillic
006	205	12/13	CYRILLIC SMALL LETTER EM	Cyrillic
006	206	12/14	CYRILLIC SMALL LETTER EN	Cyrillic
006	207	12/15	CYRILLIC SMALL LETTER O	Cyrillic
006	208	13/00	CYRILLIC SMALL LETTER PE	Cyrillic
006	209	13/01	CYRILLIC SMALL LETTER YA	Cyrillic
006	210	13/02	CYRILLIC SMALL LETTER ER	Cyrillic
006	211	13/03	CYRILLIC SMALL LETTER ES	Cyrillic
006	212	13/04	CYRILLIC SMALL LETTER TE	Cyrillic
006	213	13/05	CYRILLIC SMALL LETTER U	Cyrillic
006	214	13/06	CYRILLIC SMALL LETTER ZHE	Cyrillic
006	215	13/07	CYRILLIC SMALL LETTER VE	Cyrillic
006	216	13/08	CYRILLIC SMALL SOFT SIGN	Cyrillic
006	217	13/09	CYRILLIC SMALL LETTER YERU	Cyrillic
006	218	13/10	CYRILLIC SMALL LETTER ZE	Cyrillic
006	219	13/11	CYRILLIC SMALL LETTER SHA	Cyrillic
006	220	13/12	CYRILLIC SMALL LETTER E	Cyrillic
006	221	13/13	CYRILLIC SMALL LETTER SHCHA	Cyrillic
006	222	13/14	CYRILLIC SMALL LETTER CHE	Cyrillic
006	223	13/15	CYRILLIC SMALL HARD SIGN	Cyrillic
006	224	14/00	CYRILLIC CAPITAL LETTER YU	Cyrillic
006	225	14/01	CYRILLIC CAPITAL LETTER A	Cyrillic
006	226	14/02	CYRILLIC CAPITAL LETTER BE	Cyrillic
006	227	14/03	CYRILLIC CAPITAL LETTER TSE	Cyrillic
006	228	14/04	CYRILLIC CAPITAL LETTER DE	Cyrillic
006	229	14/05	CYRILLIC CAPITAL LETTER IE	Cyrillic
006	230	14/06	CYRILLIC CAPITAL LETTER EF	Cyrillic
006	231	14/07	CYRILLIC CAPITAL LETTER GHE	Cyrillic
006	232	14/08	CYRILLIC CAPITAL LETTER HA	Cyrillic
006	233	14/09	CYRILLIC CAPITAL LETTER I	Cyrillic
006	234	14/10	CYRILLIC CAPITAL LETTER SHORT I	Cyrillic
006	235	14/11	CYRILLIC CAPITAL LETTER KA	Cyrillic
006	236	14/12	CYRILLIC CAPITAL LETTER EL	Cyrillic
006	237	14/13	CYRILLIC CAPITAL LETTER EM	Cyrillic
006	238	14/14	CYRILLIC CAPITAL LETTER EN	Cyrillic
006	239	14/15	CYRILLIC CAPITAL LETTER O	Cyrillic
006	240	15/00	CYRILLIC CAPITAL LETTER PE	Cyrillic
006	241	15/01	CYRILLIC CAPITAL LETTER YA	Cyrillic
006	242	15/02	CYRILLIC CAPITAL LETTER ER	Cyrillic
006	243	15/03	CYRILLIC CAPITAL LETTER ES	Cyrillic
006	244	15/04	CYRILLIC CAPITAL LETTER TE	Cyrillic
006	245	15/05	CYRILLIC CAPITAL LETTER U	Cyrillic
006	246	15/06	CYRILLIC CAPITAL LETTER ZHE	Cyrillic

Byte 3	Byte 4	Code Pos	Name	Set
006	247	15/07	CYRILLIC CAPITAL LETTER VE	Cyrillic
006	248	15/08	CYRILLIC CAPITAL SOFT SIGN	Cyrillic
006	249	15/09	CYRILLIC CAPITAL LETTER YERU	Cyrillic
006	250	15/10	CYRILLIC CAPITAL LETTER ZE	Cyrillic
006	251	15/11	CYRILLIC CAPITAL LETTER SHA	Cyrillic
006	252	15/12	CYRILLIC CAPITAL LETTER E	Cyrillic
006	253	15/13	CYRILLIC CAPITAL LETTER SHCHA	Cyrillic
006	254	15/14	CYRILLIC CAPITAL LETTER CHE	Cyrillic
006	255	15/15	CYRILLIC CAPITAL HARD SIGN	Cyrillic
007	161	10/01	GREEK CAPITAL LETTER ALPHA WITH ACCENT	Greek
007	162	10/02	GREEK CAPITAL LETTER EPSILON WITH ACCENT	Greek
007	163	10/03	GREEK CAPITAL LETTER ETA WITH ACCENT	Greek
007	164	10/04	GREEK CAPITAL LETTER IOTA WITH ACCENT	Greek
007	165	10/05	GREEK CAPITAL LETTER IOTA WITH DIAERESIS	Greek
007	166	10/06	GREEK CAPITAL LETTER IOTA WITH ACCENT+DIERESIS	Greek
007	167	10/07	GREEK CAPITAL LETTER OMICRON WITH ACCENT	Greek
007	168	10/08	GREEK CAPITAL LETTER UPSILON WITH ACCENT	Greek
007	169	10/09	GREEK CAPITAL LETTER UPSILON WITH DIERESIS	Greek
007	170	10/10	GREEK CAPITAL LETTER UPSILON WITH ACCENT+DIERESIS	Greek
007	171	10/11	GREEK CAPITAL LETTER OMEGA WITH ACCENT	Greek
007	177	11/01	GREEK SMALL LETTER ALPHA WITH ACCENT	Greek
007	178	11/02	GREEK SMALL LETTER EPSILON WITH ACCENT	Greek
007	179	11/03	GREEK SMALL LETTER ETA WITH ACCENT	Greek
007	180	11/04	GREEK SMALL LETTER IOTA WITH ACCENT	Greek
007	181	11/05	GREEK SMALL LETTER IOTA WITH DIERESIS	Greek
007	182	11/06	GREEK SMALL LETTER IOTA WITH ACCENT+DIERESIS	Greek
007	183	11/07	GREEK SMALL LETTER OMICRON WITH ACCENT	Greek
007	184	11/08	GREEK SMALL LETTER UPSILON WITH ACCENT	Greek
007	185	11/09	GREEK SMALL LETTER UPSILON WITH DIERESIS	Greek
007	186	11/10	GREEK SMALL LETTER UPSILON WITH ACCENT+DIERESIS	Greek
007	187	11/11	GREEK SMALL LETTER OMEGA WITH ACCENT	Greek
007	193	12/01	GREEK CAPITAL LETTER ALPHA	Greek
007	194	12/02	GREEK CAPITAL LETTER BETA	Greek
007	195	12/03	GREEK CAPITAL LETTER GAMMA	Greek
007	196	12/04	GREEK CAPITAL LETTER DELTA	Greek
007	197	12/05	GREEK CAPITAL LETTER EPSILON	Greek
007	198	12/06	GREEK CAPITAL LETTER ZETA	Greek
007	199	12/07	GREEK CAPITAL LETTER ETA	Greek
007	200	12/08	GREEK CAPITAL LETTER THETA	Greek
007	201	12/09	GREEK CAPITAL LETTER IOTA	Greek
007	202	12/10	GREEK CAPITAL LETTER KAPPA	Greek
007	203	12/11	GREEK CAPITAL LETTER LAMBDA	Greek
007	204	12/12	GREEK CAPITAL LETTER MU	Greek
007	205	12/13	GREEK CAPITAL LETTER NU	Greek
007	206	12/14	GREEK CAPITAL LETTER XI	Greek
007	207	12/15	GREEK CAPITAL LETTER OMICRON	Greek
007	208	13/00	GREEK CAPITAL LETTER PI	Greek
007	209	13/01	GREEK CAPITAL LETTER RHO	Greek
007	210	13/02	GREEK CAPITAL LETTER SIGMA	Greek
007	212	13/04	GREEK CAPITAL LETTER TAU	Greek
007	213	13/05	GREEK CAPITAL LETTER UPSILON	Greek
007	214	13/06	GREEK CAPITAL LETTER PHI	Greek
007	215	13/07	GREEK CAPITAL LETTER CHI	Greek
007	216	13/08	GREEK CAPITAL LETTER PSI	Greek
007	217	13/09	GREEK CAPITAL LETTER OMEGA	Greek
007	225	14/01	GREEK SMALL LETTER ALPHA	Greek
007	226	14/02	GREEK SMALL LETTER BETA	Greek
007	227	14/03	GREEK SMALL LETTER GAMMA	Greek
007	228	14/04	GREEK SMALL LETTER DELTA	Greek

Byte 3	Byte 4	Code Pos	Name	Set
007	229	14/05	GREEK SMALL LETTER EPSILON	Greek
007	230	14/06	GREEK SMALL LETTER ZETA	Greek
007	231	14/07	GREEK SMALL LETTER ETA	Greek
007	232	14/08	GREEK SMALL LETTER THETA	Greek
007	233	14/09	GREEK SMALL LETTER IOTA	Greek
007	234	14/10	GREEK SMALL LETTER KAPPA	Greek
007	235	14/11	GREEK SMALL LETTER LAMBDA	Greek
007	236	14/12	GREEK SMALL LETTER MU	Greek
007	237	14/13	GREEK SMALL LETTER NU	Greek
007	238	14/14	GREEK SMALL LETTER XI	Greek
007	239	14/15	GREEK SMALL LETTER OMICRON	Greek
007	240	15/00	GREEK SMALL LETTER PI	Greek
007	241	15/01	GREEK SMALL LETTER RHO	Greek
007	242	15/02	GREEK SMALL LETTER SIGMA	Greek
007	243	15/03	GREEK SMALL LETTER FINAL SMALL SIGMA	Greek
007	244	15/04	GREEK SMALL LETTER TAU	Greek
007	245	15/05	GREEK SMALL LETTER UPSILON	Greek
007	246	15/06	GREEK SMALL LETTER PHI	Greek
007	247	15/07	GREEK SMALL LETTER CHI	Greek
007	248	15/08	GREEK SMALL LETTER PSI	Greek
007	249	15/09	GREEK SMALL LETTER OMEGA	Greek
008	161	10/01	LEFT RADICAL	Tech
008	162	10/02	TOP LEFT RADICAL	Tech
008	163	10/03	HORIZONTAL CONNECTOR	Tech
008	164	10/04	TOP INTEGRAL	Tech
008	165	10/05	BOTTOM INTEGRAL	Tech
008	166	10/06	VERTICAL CONNECTOR	Tech
008	167	10/07	TOP LEFT SQUARE BRACKET	Tech
008	168	10/08	BOTTOM LEFT SQUARE BRACKET	Tech
008	169	10/09	TOP RIGHT SQUARE BRACKET	Tech
008	170	10/10	BOTTOM RIGHT SQUARE BRACKET	Tech
008	171	10/11	TOP LEFT PARENTHESIS	Tech
008	172	10/12	BOTTOM LEFT PARENTHESIS	Tech
008	173	10/13	TOP RIGHT PARENTHESIS	Tech
008	174	10/14	BOTTOM RIGHT PARENTHESIS	Tech
008	175	10/15	LEFT MIDDLE CURLY BRACE	Tech
008	176	11/00	RIGHT MIDDLE CURLY BRACE	Tech
008	177	11/01	TOP LEFT SUMMATION	Tech
008	178	11/02	BOTTOM LEFT SUMMATION	Tech
008	179	11/03	TOP VERTICAL SUMMATION CONNECTOR	Tech
008	180	11/04	BOTTOM VERTICAL SUMMATION CONNECTOR	Tech
008	181	11/05	TOP RIGHT SUMMATION	Tech
008	182	11/06	BOTTOM RIGHT SUMMATION	Tech
008	183	11/07	RIGHT MIDDLE SUMMATION	Tech
008	188	11/12	LESS THAN OR EQUAL SIGN	Tech
008	189	11/13	NOT EQUAL SIGN	Tech
008	190	11/14	GREATER THAN OR EQUAL SIGN	Tech
008	191	11/15	INTEGRAL	Tech
008	192	12/00	THEREFORE	Tech
008	193	12/01	VARIATION, PROPORTIONAL TO	Tech
008	194	12/02	INFINITY	Tech
008	197	12/05	NABLA, DEL	Tech
008	200	12/08	IS APPROXIMATE TO	Tech
008	201	12/09	SIMILAR OR EQUAL TO	Tech
008	205	12/13	IF AND ONLY IF	Tech
008	206	12/14	IMPLIES	Tech
008	207	12/15	IDENTICAL TO	Tech
008	214	13/06	RADICAL	Tech
008	218	13/10	IS INCLUDED IN	Tech

Byte 3	Byte 4	Code Pos	Name	Set
008	219	13/11	INCLUDES	Tech
008	220	13/12	INTERSECTION	Tech
008	221	13/13	UNION	Tech
008	222	13/14	LOGICAL AND	Tech
008	223	13/15	LOGICAL OR	Tech
008	239	14/15	PARTIAL DERIVATIVE	Tech
008	246	15/06	FUNCTION	Tech
008	251	15/11	LEFT ARROW	Tech
008	252	15/12	UPWARD ARROW	Tech
008	253	15/13	RIGHT ARROW	Tech
008	254	15/14	DOWNWARD ARROW	Tech
009	223	13/15	BLANK	Special
009	224	14/00	SOLID DIAMOND	Special
009	225	14/01	CHECKERBOARD	Special
009	226	14/02	"HT"	Special
009	227	14/03	"FF"	Special
009	228	14/04	"CR"	Special
009	229	14/05	"LF"	Special
009	232	14/08	"NL"	Special
009	233	14/09	"VT"	Special
009	234	14/10	LOWER-RIGHT CORNER	Special
009	235	14/11	UPPER-RIGHT CORNER	Special
009	236	14/12	UPPER-LEFT CORNER	Special
009	237	14/13	LOWER-LEFT CORNER	Special
009	238	14/14	CROSSING-LINES	Special
009	239	14/15	HORIZONTAL LINE, SCAN 1	Special
009	240	15/00	HORIZONTAL LINE, SCAN 3	Special
009	241	15/01	HORIZONTAL LINE, SCAN 5	Special
009	242	15/02	HORIZONTAL LINE, SCAN 7	Special
009	243	15/03	HORIZONTAL LINE, SCAN 9	Special
009	244	15/04	LEFT "T"	Special
009	245	15/05	RIGHT "T"	Special
009	246	15/06	BOTTOM "T"	Special
009	247	15/07	TOP "T"	Special
009	248	15/08	VERTICAL BAR	Special
010	161	10/01	EM SPACE	Publish
010	162	10/02	EN SPACE	Publish
010	163	10/03	3/EM SPACE	Publish
010	164	10/04	4/EM SPACE	Publish
010	165	10/05	DIGIT SPACE	Publish
010	166	10/06	PUNCTUATION SPACE	Publish
010	167	10/07	THIN SPACE	Publish
010	168	10/08	HAIR SPACE	Publish
010	169	10/09	EM DASH	Publish
010	170	10/10	EN DASH	Publish
010	172	10/12	SIGNIFICANT BLANK SYMBOL	Publish
010	174	10/14	ELLIPSIS	Publish
010	175	10/15	DOUBLE BASELINE DOT	Publish
010	176	11/00	VULGAR FRACTION ONE THIRD	Publish
010	177	11/01	VULGAR FRACTION TWO THIRDS	Publish
010	178	11/02	VULGAR FRACTION ONE FIFTH	Publish
010	179	11/03	VULGAR FRACTION TWO FIFTHS	Publish
010	180	11/04	VULGAR FRACTION THREE FIFTHS	Publish
010	181	11/05	VULGAR FRACTION FOUR FIFTHS	Publish
010	182	11/06	VULGAR FRACTION ONE SIXTH	Publish
010	183	11/07	VULGAR FRACTION FIVE SIXTHS	Publish
010	184	11/08	CARE OF	Publish

Byte 3	Byte 4	Code Pos	Name	Set
010	187	11/11	FIGURE DASH	Publish
010	188	11/12	LEFT ANGLE BRACKET	Publish
010	189	11/13	DECIMAL POINT	Publish
010	190	11/14	RIGHT ANGLE BRACKET	Publish
010	191	11/15	MARKER	Publish
010	195	12/03	VULGAR FRACTION ONE EIGHTH	Publish
010	196	12/04	VULGAR FRACTION THREE EIGHTHS	Publish
010	197	12/05	VULGAR FRACTION FIVE EIGHTHS	Publish
010	198	12/06	VULGAR FRACTION SEVEN EIGHTHS	Publish
010	201	12/09	TRADEMARK SIGN	Publish
010	202	12/10	SIGNATURE MARK	Publish
010	203	12/11	TRADEMARK SIGN IN CIRCLE	Publish
010	204	12/12	LEFT OPEN TRIANGLE	Publish
010	205	12/13	RIGHT OPEN TRIANGLE	Publish
010	206	12/14	EM OPEN CIRCLE	Publish
010	207	12/15	EM OPEN RECTANGLE	Publish
010	208	13/00	LEFT SINGLE QUOTATION MARK	Publish
010	209	13/01	RIGHT SINGLE QUOTATION MARK	Publish
010	210	13/02	LEFT DOUBLE QUOTATION MARK	Publish
010	211	13/03	RIGHT DOUBLE QUOTATION MARK	Publish
010	212	13/04	PRESCRIPTION, TAKE, RECIPE	Publish
010	214	13/06	MINUTES	Publish
010	215	13/07	SECONDS	Publish
010	217	13/09	LATIN CROSS	Publish
010	218	13/10	HEXAGRAM	Publish
010	219	13/11	FILLED RECTANGLE BULLET	Publish
010	220	13/12	FILLED LEFT TRIANGLE BULLET	Publish
010	221	13/13	FILLED RIGHT TRIANGLE BULLET	Publish
010	222	13/14	EM FILLED CIRCLE	Publish
010	223	13/15	EM FILLED RECTANGLE	Publish
010	224	14/00	EN OPEN CIRCLE BULLET	Publish
010	225	14/01	EN OPEN SQUARE BULLET	Publish
010	226	14/02	OPEN RECTANGULAR BULLET	Publish
010	227	14/03	OPEN TRIANGULAR BULLET UP	Publish
010	228	14/04	OPEN TRIANGULAR BULLET DOWN	Publish
010	229	14/05	OPEN STAR	Publish
010	230	14/06	EN FILLED CIRCLE BULLET	Publish
010	231	14/07	EN FILLED SQUARE BULLET	Publish
010	232	14/08	FILLED TRIANGULAR BULLET UP	Publish
010	233	14/09	FILLED TRIANGULAR BULLET DOWN	Publish
010	234	14/10	LEFT POINTER	Publish
010	235	14/11	RIGHT POINTER	Publish
010	236	14/12	CLUB	Publish
010	237	14/13	DIAMOND	Publish
010	238	14/14	HEART	Publish
010	240	15/00	MALTESE CROSS	Publish
010	241	15/01	DAGGER	Publish
010	242	15/02	DOUBLE DAGGER	Publish
010	243	15/03	CHECK MARK, TICK	Publish
010	244	15/04	BALLOT CROSS	Publish
010	245	15/05	MUSICAL SHARP	Publish
010	246	15/06	MUSICAL FLAT	Publish
010	247	15/07	MALE SYMBOL	Publish
010	248	15/08	FEMALE SYMBOL	Publish
010	249	15/09	TELEPHONE SYMBOL	Publish
010	250	15/10	TELEPHONE RECORDER SYMBOL	Publish
010	251	15/11	PHONOGRAPH COPYRIGHT SIGN	Publish
010	252	15/12	CARET	Publish
010	253	15/13	SINGLE LOW QUOTATION MARK	Publish
010	254	15/14	DOUBLE LOW QUOTATION MARK	Publish
010	255	15/15	CURSOR	Publish

Byte 3	Byte 4	Code Pos	Name	Set
011	163	10/03	LEFT CARET	APL
011	166	10/06	RIGHT CARET	APL
011	168	10/08	DOWN CARET	APL
011	169	10/09	UP CARET	APL
011	192	12/00	OVERBAR	APL
011	194	12/02	DOWN TACK	APL
011	195	12/03	UP SHOE (CAP)	APL
011	196	12/04	DOWN STILE	APL
011	198	12/06	UNDERBAR	APL
011	202	12/10	JOT	APL
011	204	12/12	QUAD	APL
011	206	12/14	UP TACK	APL
011	207	12/15	CIRCLE	APL
011	211	13/03	UP STILE	APL
011	214	13/06	DOWN SHOE (CUP)	APL
011	216	13/08	RIGHT SHOE	APL
011	218	13/10	LEFT SHOE	APL
011	220	13/12	LEFT TACK	APL
011	252	15/12	RIGHT TACK	APL
012	224	14/00	HEBREW LETTER ALEPH	Hebrew
012	225	14/01	HEBREW LETTER BETH	Hebrew
012	226	14/02	HEBREW LETTER GIMMEL	Hebrew
012	227	14/03	HEBREW LETTER DALETH	Hebrew
012	228	14/04	HEBREW LETTER HE	Hebrew
012	229	14/05	HEBREW LETTER WAW	Hebrew
012	230	14/06	HEBREW LETTER ZAYIN	Hebrew
012	231	14/07	HEBREW LETTER HET	Hebrew
012	232	14/08	HEBREW LETTER TETH	Hebrew
012	233	14/09	HEBREW LETTER YOD	Hebrew
012	234	14/10	HEBREW LETTER FINAL KAPH	Hebrew
012	235	14/11	HEBREW LETTER KAPH	Hebrew
012	236	14/12	HEBREW LETTER LAMED	Hebrew
012	237	14/13	HEBREW LETTER FINAL MEM	Hebrew
012	238	14/14	HEBREW LETTER MEM	Hebrew
012	239	14/15	HEBREW LETTER FINAL NUN	Hebrew
012	240	15/00	HEBREW LETTER NUN	Hebrew
012	241	15/01	HEBREW LETTER SAMEKH	Hebrew
012	242	15/02	HEBREW LETTER A'YIN	Hebrew
012	243	15/03	HEBREW LETTER FINAL PE	Hebrew
012	244	15/04	HEBREW LETTER PE	Hebrew
012	245	15/05	HEBREW LETTER FINAL ZADI	Hebrew
012	246	15/06	HEBREW LETTER ZADI	Hebrew
012	247	15/07	HEBREW KUF	Hebrew
012	248	15/08	HEBREW RESH	Hebrew
012	249	15/09	HEBREW SHIN	Hebrew
012	250	15/10	HEBREW TAF	Hebrew
255	008	00/08	BACKSPACE, BACK SPACE, BACK CHAR	Keyboard
255	009	00/09	TAB	Keyboard
255	010	00/10	LINEFEED, LF	Keyboard
255	011	00/11	CLEAR	Keyboard
255	013	00/13	RETURN, ENTER	Keyboard
255	019	01/03	PAUSE, HOLD, SCROLL LOCK	Keyboard
255	027	01/11	ESCAPE	Keyboard
255	032	02/00	MULTI-KEY CHARACTER PREFACE	Keyboard
255	033	02/01	KANJI, KANJI CONVERT	Keyboard
255	080	05/00	HOME	Keyboard
255	081	05/01	LEFT, MOVE LEFT, LEFT ARROW	Keyboard

Byte 3	Byte 4	Code Pos	Name	Set
255	082	05/02	UP, MOVE UP, UP ARROW	Keyboard
255	083	05/03	RIGHT, MOVE RIGHT, RIGHT ARROW	Keyboard
255	084	05/04	DOWN, MOVE DOWN, DOWN ARROW	Keyboard
255	085	05/05	PRIOR, PREVIOUS	Keyboard
255	086	05/06	NEXT	Keyboard
255	087	05/07	END, EOL	Keyboard
255	088	05/08	BEGIN, BOL	Keyboard
255	096	06/00	SELECT, MARK	Keyboard
255	097	06/01	PRINT	Keyboard
255	098	06/02	EXECUTE, RUN, DO	Keyboard
255	099	06/03	INSERT, INSERT HERE	Keyboard
255	101	06/05	UNDO, OOPS	Keyboard
255	102	06/06	REDO, AGAIN	Keyboard
255	103	06/07	MENU	Keyboard
255	104	06/08	FIND, SEARCH	Keyboard
255	105	06/09	CANCEL, STOP, ABORT, EXIT	Keyboard
255	106	06/10	HELP, QUESTION MARK	Keyboard
255	107	06/11	BREAK	Keyboard
255	126	07/14	MODE SWITCH, SCRIPT SWITCH, CHARACTER SET SWITCH	Keyboard
255	127	07/15	NUM LOCK	Keyboard
255	128	08/00	KEYPAD SPACE	Keyboard
255	137	08/09	KEYPAD TAB	Keyboard
255	141	08/13	KEYPAD ENTER	Keyboard
255	145	09/01	KEYPAD F1, PF1, A	Keyboard
255	146	09/02	KEYPAD F2, PF2, B	Keyboard
255	147	09/03	KEYPAD F3, PF3, C	Keyboard
255	148	09/04	KEYPAD F4, PF4, D	Keyboard
255	170	10/10	KEYPAD MULTIPLICATION SIGN, ASTERISK	Keyboard
255	171	10/11	KEYPAD PLUS SIGN	Keyboard
255	172	10/12	KEYPAD SEPARATOR, COMMA	Keyboard
255	173	10/13	KEYPAD MINUS SIGN, HYPHEN	Keyboard
255	174	10/14	KEYPAD DECIMAL POINT, FULL STOP	Keyboard
255	175	10/15	KEYPAD DIVISION SIGN, SOLIDUS	Keyboard
255	176	11/00	KEYPAD DIGIT ZERO	Keyboard
255	177	11/01	KEYPAD DIGIT ONE	Keyboard
255	178	11/02	KEYPAD DIGIT TWO	Keyboard
255	179	11/03	KEYPAD DIGIT THREE	Keyboard
255	180	11/04	KEYPAD DIGIT FOUR	Keyboard
255	181	11/05	KEYPAD DIGIT FIVE	Keyboard
255	182	11/06	KEYPAD DIGIT SIX	Keyboard
255	183	11/07	KEYPAD DIGIT SEVEN	Keyboard
255	184	11/08	KEYPAD DIGIT EIGHT	Keyboard
255	185	11/09	KEYPAD DIGIT NINE	Keyboard
255	189	11/13	KEYPAD EQUALS SIGN	Keyboard
255	190	11/14	F1	Keyboard
255	191	11/15	F2	Keyboard
255	192	12/00	F3	Keyboard
255	193	12/01	F4	Keyboard
255	194	12/02	F5	Keyboard
255	195	12/03	F6	Keyboard
255	196	12/04	F7	Keyboard
255	197	12/05	F8	Keyboard
255	198	12/06	F9	Keyboard
255	199	12/07	F10	Keyboard
255	200	12/08	F11, L1	Keyboard
255	201	12/09	F12, L2	Keyboard
255	202	12/10	F13, L3	Keyboard
255	203	12/11	F14, L4	Keyboard
255	204	12/12	F15, L5	Keyboard
255	205	12/13	F16, L6	Keyboard
255	206	12/14	F17, L7	Keyboard

Byte 3	Byte 4	Code Pos	Name	Set
255	207	12/15	F18, L8	Keyboard
255	208	13/00	F19, L9	Keyboard
255	209	13/01	F20, L10	Keyboard
255	210	13/02	F21, R1	Keyboard
255	211	13/03	F22, R2	Keyboard
255	212	13/04	F23, R3	Keyboard
255	213	13/05	F24, R4	Keyboard
255	214	13/06	F25, R5	Keyboard
255	215	13/07	F26, R6	Keyboard
255	216	13/08	F27, R7	Keyboard
255	217	13/09	F28, R8	Keyboard
255	218	13/10	F29, R9	Keyboard
255	219	13/11	F30, R10	Keyboard
255	220	13/12	F31, R11	Keyboard
255	221	13/13	F32, R12	Keyboard
255	222	13/14	F33, R13	Keyboard
255	223	13/15	F34, R14	Keyboard
255	224	14/00	F35, R15	Keyboard
255	225	14/01	LEFT SHIFT	Keyboard
255	226	14/02	RIGHT SHIFT	Keyboard
255	227	14/03	LEFT CONTROL	Keyboard
255	228	14/04	RIGHT CONTROL	Keyboard
255	229	14/05	CAPS LOCK	Keyboard
255	230	14/06	SHIFT LOCK	Keyboard
255	231	14/07	LEFT META	Keyboard
255	232	14/08	RIGHT META	Keyboard
255	233	14/09	LEFT ALT	Keyboard
255	234	14/10	RIGHT ALT	Keyboard
255	235	14/11	LEFT SUPER	Keyboard
255	236	14/12	RIGHT SUPER	Keyboard
255	237	14/13	LEFT HYPER	Keyboard
255	238	14/14	RIGHT HYPER	Keyboard
255	255	15/15	DELETE, RUBOUT	Keyboard

Appendix B
Protocol Encoding

The sections in this appendix correspond to their number counterparts in the protocol document.

Syntax

All numbers are in decimal, unless prefixed with #x, in which case they are in hexadecimal (base 16).

The general syntax used to describe requests, replies, errors, events, and compound types is:

```
NameofThing
 encode-form
 ...
 encode-form
```

Each encode-form describes a single component.

For components described in the protocol document as:

```
name: TYPE
```

The encode-form is:

```
N               TYPE    name
```

N is the number of bytes occupied in the data stream, and TYPE is the interpretation of those bytes. For example,

```
depth: CARD8
```

becomes:

```
1               CARD8   depth
```

For components with a static numeric value the encode-form is:

```
N               value   name
```

The value is always interpreted as an N-byte unsigned integer. For example, the first two bytes of a Window error are always 0 (indicating an error in general) and 3 (indicating the Window error in particular):

```
1               0       Error
1               3       code
```

For components described in the protocol document as:

```
name: { Name1,..., NameN }
```

The encode-form is:

```
N                       name
                value1 Name1
                ...
                valueN NameN
```

The value is always interpreted as an N-byte unsigned integer. Note that the size of N is sometimes larger that than strictly required to encode the values. For example,

```
class: { InputOutput, InputOnly, CopyFromParent }
```

Becomes:

```
2                                 class
                0       CopyFromParent
                1       InputOutput
                2       InputOnly
```

For components described in the protocol document as:

```
NAME: TYPE or Alternative1...or AlternativeN
```

The encode-form is

```
     N                TYPE                  NAME
                      value1   Alternative1
                      ...
                      valueI   AlternativeI
```

The alternative values are guaranteed not to conflict with the encoding of TYPE. For example,

destination: WINDOW or PointerWindow or InputFocus

becomes:

```
     4                WINDOW                destination
                      0        PointerWindow
                      1        InputFocus
```

For components described in the protocol document as

value-mask: BITMASK

The encode-form is

```
     N                BITMASK               value-mask
                      mask1    mask-name1
                      ...
                      maskI    mask-nameI
```

The individual bits in the mask are specified and named, and N is 2 or 4. The most significant bit in a BITMASK is "reserved" for use in defining chained (multi-word) bitmasks, as extensions augment existing core requests. The precise interpretation of this bit is not yet defined here, although a probable mechanism is that a 1-bit indicates that another N bytes of bitmask follows, with bits within the overall mask still interpreted from least to most significant with an N-byte unit, with N-byte units interpreted in stream order, and with the overall mask being byte-swapped in individual N-byte units.

For LISTofVALUE encodings, the request is followed by a section of the form:

```
     VALUEs
      encode-form
      ...
      encode-form
```

Listing an encode-form for each VALUE. The NAME in each encode-form keys to the corresponding BITMASK bit. The encoding of a VALUE always occupies 4 bytes, but the number of bytes specified in the encoding-form indicates how many of the (least significant) bytes are actually used; the remaining bytes are unused and their values do not matter.

In various cases the number of bytes occupied by a component will be specified by a lowercase single-letter variable name instead of a specific numeric value, and often some other component will have its value specified as a simple numeric expression involving these variables. Components specified with such expressions are always interpret as unsigned integers. The "scope" of such variables is always just the enclosing request, reply, error, event, or compound type structure. For example:

```
     2                3+n                   request length
     4n               LISTofPOINT           points
```

For unused bytes (the values of the bytes are undefined and do no matter), the encode-form is:

```
     N                              unused
```

If the number of unused bytes is variable, the encode-form typically is:

```
     p                              unused, p=pad(E)
```

Where E is some expression. pad(E) is the number (of bytes) needed to round E up to a multiple of 4:

pad(E) = (4 - (E mod 4)) mod 4

Common Types

LISTofFOO

In this document the LISTof notation strictly means some number of repetitions of the FOO encoding; the actual length of the list is encoded elsewhere.

SETofFOO

A set is always represented by a bitmask, with a 1-bit indicating presence in the set.

BITMASK: CARD32

WINDOW: CARD32

PIXMAP: CARD32

CURSOR: CARD32

FONT: CARD32

GCONTEXT: CARD32

COLORMAP: CARD32

DRAWABLE: CARD32

FONTABLE: CARD32

ATOM: CARD32

VISUALID: CARD32

BYTE: 8-bit value

INT8: 8-bit signed integer

INT16: 16-bit signed integer

INT32: 32-bit signed integer

CARD8: 8-bit unsigned integer

CARD16: 16-bit unsigned integer

CARD32: 32-bit unsigned integer

TIMESTAMP: CARD32

BITGRAVITY

0	Forget
1	NorthWest
2	North
3	NorthEast
4	West
5	Center
6	East
7	SouthWest
8	South
9	SouthEast
10	Static

WINGRAVITY

0	Unmap
1	NorthWest
2	North
3	NorthEast
4	West
5	Center
6	East
7	SouthWest
8	South
9	SouthEast
10	Static

BOOL

0	False
1	True

SETofEVENT

#x00000001	KeyPress
#x00000002	KeyRelease
#x00000004	ButtonPress
#x00000008	ButtonRelease
#x00000010	EnterWindow
#x00000020	LeaveWindow

#x00000040	PointerMotion
#x00000080	PointerMotionHint
#x00000100	Button1Motion
#x00000200	Button2Motion
#x00000400	Button3Motion
#x00000800	Button4Motion
#x00001000	Button5Motion
#x00002000	ButtonMotion
#x00004000	KeymapState
#x00008000	Exposure
#x00010000	VisibilityChange
#x00020000	StructureNotify
#x00040000	ResizeRedirect
#x00080000	SubstructureNotify
#x00100000	SubstructureRedirect
#x00200000	FocusChange
#x00400000	PropertyChange
#x00800000	ColormapChange
#x01000000	OwnerGrabButton
#xfe000000	unused but must be zero

SETofPOINTEREVENT
encodings are the same as for SETofEVENT, except with

#xffff8003	unused but must be zero

SETofDEVICEEVENT
encodings are the same as for SETofEVENT, except with

#xffffc0b0	unused but must be zero

KEYSYM: CARD32

KEYCODE: CARD8

BUTTON: CARD8

SETofKEYBUTMASK

#x0001	Shift
#x0002	Lock
#x0004	Control
#x0008	Mod1
#x0010	Mod2
#x0020	Mod3
#x0040	Mod4
#x0080	Mod5
#x0100	Button1
#x0200	Button2
#x0400	Button3
#x0800	Button4
#x1000	Button5
#xc000	unused but must be zero

SETofKEYMASK
encodings are the same as for KEYBUTMASK, except with

#xff00	unused but must be zero

STRING8: LISTofCARD8

STRING16: LISTofCHAR2B

CHAR2B

1	CARD8	byte1
1	CARD8	byte2

POINT

2	INT16	x
2	INT16	y

RECTANGLE

2	INT16	x

2	INT16	y	
2	CARD16	width	
2	CARD16	height	

ARC

2	INT16	x	
2	INT16	y	
2	CARD16	width	
2	CARD16	height	
2	INT16	angle1	
2	INT16	angle2	

HOST

1			family
	0	Internet	
	1	DECnet	
	2	Chaos	
1			unused
2	n		length of address
n	LISTofBYTE		address
p			unused, p=pad(n)

STR

1	n	length of name in bytes
n	STRING8	name

Errors

Request

1	0	Error
1	1	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Value

1	0	Error
1	2	code
2	CARD16	sequence number
4	<32-bits>	bad value
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Window

1	0	Error
1	3	code
2	CARD16	sequence number
4	CARD32	bad resource id
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Pixmap

1	0	Error
1	4	code
2	CARD16	sequence number
4	CARD32	bad resource id
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Atom

1	0	Error

1	5	code
2	CARD16	sequence number
4	CARD32	bad atom id
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Cursor

1	0	Error
1	6	code
2	CARD16	sequence number
4	CARD32	bad resource id
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Font

1	0	Error
1	7	code
2	CARD16	sequence number
4	CARD32	bad resource id
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Match

1	0	Error
1	8	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Drawable

1	0	Error
1	9	code
2	CARD16	sequence number
4	CARD32	bad resource id
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Access

1	0	Error
1	10	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Alloc

1	0	Error
1	11	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Colormap

1	0	Error
1	12	code
2	CARD16	sequence number
4	CARD32	bad resource id

```
    2     CARD16                         minor opcode
    1     CARD8                          major opcode
    21                                   unused

GContext
    1     0                              Error
    1     13                             code
    2     CARD16                         sequence number
    4     CARD32                         bad resource id
    2     CARD16                         minor opcode
    1     CARD8                          major opcode
    21                                   unused

IDChoice
    1     0                              Error
    1     14                             code
    2     CARD16                         sequence number
    4     CARD32                         bad resource id
    2     CARD16                         minor opcode
    1     CARD8                          major opcode
    21                                   unused

Name
    1     0                              Error
    1     15                             code
    2     CARD16                         sequence number
    4                                    unused
    2     CARD16                         minor opcode
    1     CARD8                          major opcode
    21                                   unused

Length
    1     0                              Error
    1     16                             code
    2     CARD16                         sequence number
    4                                    unused
    2     CARD16                         minor opcode
    1     CARD8                          major opcode
    21                                   unused

Implementation
    1     0                              Error
    1     17                             code
    2     CARD16                         sequence number
    4                                    unused
    2     CARD16                         minor opcode
    1     CARD8                          major opcode
    21                                   unused
```

Keyboards

KEYCODE values are always greater than 7 (and less than 256).

KEYSYMs with the bit #x10000000 set are reserved as "vendor-specific".

The names and encodings of the standard KEYSYMs are contained in Appendix B, KEYSYM Encoding.

Pointers

BUTTON values are numbered starting with one.

Predefined Atoms

PRIMARY	1	WM_NORMAL_HINTS	40
SECONDARY	2	WM_SIZE_HINTS	41
ARC	3	WM_ZOOM_HINTS	42
ATOM	4	MIN_SPACE	43
BITMAP	5	NORM_SPACE	44

CARDINAL	6	MAX_SPACE	45
COLORMAP	7	END_SPACE	46
CURSOR	8	SUPERSCRIPT_X	47
CUT_BUFFER0	9	SUPERSCRIPT_Y	48
CUT_BUFFER1	10	SUBSCRIPT_X	49
CUT_BUFFER2	11	SUBSCRIPT_Y	50
CUT_BUFFER3	12	UNDERLINE_POSITION	51
CUT_BUFFER4	13	UNDERLINE_THICKNESS	52
CUT_BUFFER5	14	STRIKEOUT_ASCENT	53
CUT_BUFFER6	15	STRIKEOUT_DESCENT	54
CUT_BUFFER7	16	ITALIC_ANGLE	55
DRAWABLE	17	X_HEIGHT	56
FONT	18	QUAD_WIDTH	57
INTEGER	19	WEIGHT	58
PIXMAP	20	POINT_SIZE	59
POINT	21	RESOLUTION	60
RECTANGLE	22	COPYRIGHT	61
RESOURCE_MANGER	23	NOTICE	62
RGB_COLOR_MAP	24	FONT_NAME	63
RGB_BEST_MAP	25	FAMILY_NAME	64
RGB_BLUE_MAP	26	FULL_NAME	65
RGB_DEFAULT_MAP	27	CAP_HEIGHT	66
RGB_GRAY_MAP	28	WM_CLASS	67
RGB_GREEN_MAP	29	WM_TRANSIENT_FOR	68
RGB_RED_MAP	30		
STRING	31		
VISUALID	32		
WINDOW	33		
WM_COMMAND	34		
WM_HINTS	35		
WM_CLIENT_MACHINE	36		
WM_ICON_NAME	37		
WM_ICON_SIZE	38		
WM_NAME	39		

Connection Setup

For TCP connections, displays on a given host are numbered starting from 0, and the server for display N listens and accepts connections on port 6000+N.

Information sent by the client at connection setup:

```
1                                 byte-order
     #x42          MSB first
     #x6C          LSB first
1                                 unused
2    CARD16                       protocol-major-version
2    CARD16                       protocol-minor-version
2    n                            length of authorization-protocol-name
2    d                            length of authorization-protocol-data
2                                 unused
n    STRING8                      authorization-protocol-name
p                                 unused, p=pad(n)
d    STRING8                      authorization-protocol-data
q                                 unused, q=pad(d)
```

Except where explicitly noted in the protocol, all 16-bit and 32-bit quantities sent by the client must be transmitted with the specified byte order, and all 16-bit and 32-bit quantities returned by the server will be transmitted with this byte order.

Information received by the client if authorization fails:

```
1    0                            failed
1    n                            length of reason in bytes
2    CARD16                       protocol-major-version
2    CARD16                       protocol-minor-version
2    (n+p)/4                      length in 4-byte units of "additional data"
n    STRING8                      reason
```

p			unused, p=pad(n)

Information received by the client if authorization is accepted:

1	1		success
1			unused
2	CARD16		protocol-major-version
2	CARD16		protocol-minor-version
2	8+2n+(v+p+m)/4		length in 4-byte units of "additional data"
4	CARD32		release-number
4	CARD32		resource-id-base
4	CARD32		resource-id-mask
4	CARD32		motion-buffer-size
2	v		length of vendor
2	CARD16		maximum-request-length
1	CARD8		number of SCREENs in roots
1	n		number for FORMATs in pixmap-formats
1			image-byte-order
	0	LSBFirst	
	1	MSBFirst	
1			bitmap-format-bit-order
	0	LeastSignificant	
	1	MostSignificant	
1	CARD8		bitmap-format-scanline-unit
1	CARD8		bitmap-format-scanline-pad
1	KEYCODE		min-keycode
1	KEYCODE		max-keycode
4			unused
v	STRING8		vendor
p			unused, p=pad(v)
8n	LISTofFORMAT		pixmap-formats
m	LISTofSCREEN		roots (m is always a multiple of 4)

FORMAT

1	CARD8		depth
1	CARD8		bits-per-pixel
1	CARD8		scanline-pad
5			unused

SCREEN

4	WINDOW		root
4	COLORMAP		default-colormap
4	CARD32		white-pixel
4	CARD32		black-pixel
4	SETofEVENT		current-input-masks
2	CARD16		width-in-pixels
2	CARD16		height-in-pixels
2	CARD16		width-in-millimeters
2	CARD16		height-in-millimeters
2	CARD16		min-installed-maps
2	CARD16		max-installed-maps
4	VISUALID		root-visual
1			backing-stores
	0	Never	
	1	WhenMapped	
	2	Always	
1	BOOL		save-unders
1	CARD8		root-depth
1	CARD8		number of DEPTHs in allowed-depths
n	LISTofDEPTH		allowed-depths (n is always a multiple of 4)

DEPTH

1	CARD8		depth
1			unused
2	n		number of VISUALTYPES in visuals
4			unused

24n	LISTofVISUALTYPE		visuals

VISUALTYPE

4	VISUALID		visual-id
1			class
	0	StaticGray	
	1	GrayScale	
	2	StaticColor	
	3	PseudoColor	
	4	TrueColor	
	5	DirectColor	
1	CARD8		bits-per-rgb-value
2	CARD16		colormap-entries
4	CARD32		red-mask
4	CARD32		green-mask
4	CARD32		blue-mask
4			unused

Requests

CreateWindow

1	1		opcode
1	CARD8		depth
2	8+n		request length
4	WINDOW		wid
4	WINDOW		parent
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
2			class
	0	CopyFromParent	
	1	InputOutput	
	2	InputOnly	
4	VISUALID		visual
	0	CopyFromParent	
4	BITMASK		value-mask (has n 1-bits)
	#x00000001	background-pixmap	
	#x00000002	background-pixel	
	#x00000004	border-pixmap	
	#x00000008	border-pixel	
	#x00000010	bit-gravity	
	#x00000020	win-gravity	
	#x00000040	backing-store	
	#x00000080	backing-planes	
	#x00000100	backing-pixel	
	#x00000200	override-redirect	
	#x00000400	save-under	
	#x00000800	event-mask	
	#x00001000	do-not-propagate-mask	
	#x00002000	colormap	
	#x00004000	cursor	
4n	LISTofVALUE		value-list

VALUEs

4	PIXMAP		background-pixmap
	0	None	
	1	ParentRelative	
4	CARD32		background-pixel
4	PIXMAP		border-pixmap
	0	CopyFromParent	
4	CARD32		border-pixel
1	BITGRAVITY		bit-gravity
1	WINGRAVITY		win-gravity
1			backing-store

	0	NotUseful	
	1	WhenMapped	
	2	Always	
4	CARD32		backing-planes
4	CARD32		backing-pixel
1	BOOL		override-redirect
1	BOOL		save-under
4	SETofEVENT		event-mask
4	SETofDEVICEEVENT		do-not-propagate-mask
4	COLORMAP		colormap
	0	CopyFromParent	
4	CURSOR		cursor
	0	None	

ChangeWindowAttributes

1	2		opcode
1			unused
2	3+n		request length
4	WINDOW		window
4	BITMASK		value-mask (has n 1-bits)
	encodings are the same as for CreateWindow		
4n	LISTofVALUE		value-list
	encodings are the same as for CreateWindow		

GetWindowAttributes

1	3		opcode
1			unused
2	2		request length
4	WINDOW		window

=>

1	1		Reply
1			backing-store
	0	NotUseful	
	1	WhenMapped	
	2	Always	
2	CARD16		sequence number
4	3		reply length
4	VISUALID		visual
2			class
	1	InputOutput	
	2	InputOnly	
1	BITGRAVITY		bit-gravity
1	WINGRAVITY		win-gravity
4	CARD32		backing-planes
4	CARD32		backing-pixel
1	BOOL		save-under
1	BOOL		map-is-installed
1			map-state
	0	Unmapped	
	1	Unviewable	
	2	Viewable	
1	BOOL		override-redirect
4	COLORMAP		colormap
	0	None	
4	SETofEVENT		all-event-masks
4	SETofEVENT		your-event-mask
2	SETofDEVICEEVENT		do-not-propagate-mask
2			unused

DestroyWindow

1	4		opcode
1			unused
2	2		request length
4	WINDOW		window

DestroySubwindows

1	5		opcode
1			unused
2	2		request length
4	WINDOW		window

ChangeSaveSet

1	6		opcode
1			mode
	0	Insert	
	1	Delete	
2	2		request length
4	WINDOW		window

ReparentWindow

1	7		opcode
1			unused
2	4		request length
4	WINDOW		window
4	WINDOW		parent
2	INT16		x
2	INT16		y

MapWindow

1	8		opcode
1			unused
2	2		request length
4	WINDOW		window

MapSubwindows

1	9		opcode
1			unused
2	2		request length
4	WINDOW		window

UnmapWindow

1	10		opcode
1			unused
2	2		request length
4	WINDOW		window

UnmapSubwindows

1	11		opcode
1			unused
2	2		request length
4	WINDOW		window

ConfigureWindow

1	12		opcode
1			unused
2	3+n		request length
4	WINDOW		window
2	BITMASK		value-mask (has n 1-bits)
	#x0001	x	
	#x0002	y	
	#x0004	width	
	#x0008	height	
	#x0010	border-width	
	#x0020	sibling	
	#x0040	stack-mode	
2			unused
4n	LISTofVALUE		value-list

VALUEs

2	INT16		x
2	INT16		y

2	CARD16		width
2	CARD16		height
2	CARD16		border-width
4	WINDOW		sibling
1			stack-mode
	0	Above	
	1	Below	
	2	TopIf	
	3	BottomIf	
	4	Opposite	

CirculateWindow

1	13		opcode
1			direction
	0	RaiseLowest	
	1	LowerHighest	
2	2		request length
4	WINDOW		window

GetGeometry

1	14		opcode
1			unused
2	2		request length
4	DRAWABLE		drawable

=>

1	1		Reply
1	CARD8		depth
2	CARD16		sequence number
4	0		reply length
4	WINDOW		root
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
10			unused

QueryTree

1	15		opcode
1			unused
2	2		request length
4	WINDOW		window

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	n		reply length
4	WINDOW		root
4	WINDOW		parent
	0	None	
2	n		number of WINDOWs in children
14			unused
4n	LISTofWINDOW		children

InternAtom

1	16		opcode
1	BOOL		only-if-exists
2	2+(n+p)/4		request length
2	n		length of name
2			unused
n	STRING8		name
p			unused, p=pad(n)

=>

1	1		Reply

1			unused
2	CARD16		sequence number
4	0		reply length
4	ATOM		atom
	0	None	
20			unused

GetAtomName

1	17		opcode
1			unused
2	2		request length
4	ATOM		atom

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	(n+p)/4		reply length
2	n		length of name
22			unused
n			name
p			unused, p=pad(n)

ChangeProperty

1	18		opcode
1			mode
	0	Replace	
	1	Prepend	
	2	Append	
2	6+(n+p)/4		request length
4	WINDOW		window
4	ATOM		property
4	ATOM		type
1	CARD8		format
3			unused
4	CARD32		length of data in format units
			(= n for format = 8)
			(= n/2 for format = 16)
			(= n/4 for format = 32)
n	LISTofBYTE		data
			(n is a multiple of 2 for format = 16)
			(n is a multiple of 4 for format = 32)
p			unused, p=pad(n)

DeleteProperty

1	19		opcode
1			unused
2	3		request length
4	WINDOW		window
4	ATOM		property

GetProperty

1	20		opcode
1	BOOL		delete
2	6		request length
4	WINDOW		window
4	ATOM		property
4	ATOM		type
	0	AnyPropertyType	
4	CARD32		long-offset
4	CARD32		long-length

=>

1	1		Reply
1	CARD8		format
2	CARD16		sequence number

4	(n+p)/4		reply length
4	ATOM		type
	0	None	
4	CARD32		bytes-after
4	CARD32		length of value in format units (= 0 for format = 0) (= n for format = 8) (= n/2 for format = 16) (= n/4 for format = 32)
12			unused
n	LISTofBYTE		value (n is zero for format = 0) (n is a multiple of 2 for format = 16) (n is a multiple of 4 for format = 32)
p			unused, p=pad(n)

ListProperties

1	21		opcode
1			unused
2	2		request length
4	WINDOW		window

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	n		reply length
2	n		number of ATOMs in atoms
22			unused
4n	LISTofATOM		atoms

SetSelectionOwner

1	22		opcode
1			unused
2	4		request length
4	WINDOW		owner
	0	None	
4	ATOM		selection
4	TIMESTAMP		time
	0	CurrentTime	

GetSelectionOwner

1	23		opcode
1			unused
2	2		request length
4	ATOM		selection

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
4	WINDOW		owner
	0	None	
20			unused

ConvertSelection

1	24		opcode
1			unused
2	6		request length
4	WINDOW		requestor
4	ATOM		selection
4	ATOM		target
4	ATOM		property
	0	None	
4	TIMESTAMP		time

	0	CurrentTime	

SendEvent

1	25		opcode
1	BOOL		propagate
2	11		request length
4	WINDOW		destination
	0	PointerWindow	
	1	InputFocus	
4	SETofEVENT		event-mask
32			event
	standard event format (see the Events section)		

GrabPointer

1	26		opcode
1	BOOL		owner-events
2	6		request length
4	WINDOW		grab-window
2	SETofPOINTEREVENT		event-mask
1			pointer-mode
	0	Synchronous	
	1	Asynchronous	
1			keyboard-mode
	0	Synchronous	
	1	Asynchronous	
4	WINDOW		confine-to
	0	None	
4	CURSOR		cursor
	0	None	
4	TIMESTAMP		time
	0	CurrentTime	

=>

1	1		Reply
1			status
	0	Success	
	1	AlreadyGrabbed	
	2	InvalidTime	
	3	NotViewable	
	4	Frozen	
2	CARD16		sequence number
4	0		reply length
24			unused

UngrabPointer

1	27		opcode
1			unused
2	2		request length
4	TIMESTAMP		time
	0	CurrentTime	

GrabButton

1	28		opcode
1	BOOL		owner-events
2	6		request length
4	WINDOW		grab-window
2	SETofPOINTEREVENT		event-mask
1			pointer-mode
	0	Synchronous	
	1	Asynchronous	
1			keyboard-mode
	0	Synchronous	
	1	Asynchronous	
4	WINDOW		confine-to
	0	None	
4	CURSOR		cursor

	0	None	
1	BUTTON		button
	0	AnyButton	
1			unused
2	SETofKEYMASK		modifiers
	#x8000	AnyModifier	

UngrabButton

1	29		opcode
1	BUTTON		button
	0	AnyButton	
2	3		request length
4	WINDOW		grab-window
2	SETofKEYMASK		modifiers
	#x8000	AnyModifier	
2			unused

ChangeActivePointerGrab

1	30		opcode
1			unused
2	4		request length
4	CURSOR		cursor
	0	None	
4	TIMESTAMP		time
	0	CurrentTime	
2	SETofPOINTEREVENT		event-mask
2			unused

GrabKeyboard

1	31		opcode
1	BOOL		owner-events
2	4		request length
4	WINDOW		grab-window
4	TIMESTAMP		time
	0	CurrentTime	
1			pointer-mode
	0	Synchronous	
	1	Asynchronous	
1			keyboard-mode
	0	Synchronous	
	1	Asynchronous	
2			unused

=>

1	1		Reply
1			status
	0	Success	
	1	AlreadyGrabbed	
	2	InvalidTime	
	3	NotViewable	
	4	Frozen	
2	CARD16		sequence number
4	0		reply length
24			unused

UngrabKeyboard

1	32		opcode
1			unused
2	2		request length
4	TIMESTAMP		time
	0	CurrentTime	

GrabKey

1	33		opcode
1	BOOL		owner-events
2	4		request length

4	WINDOW		grab-window
2	SETofKEYMASK		modifiers
	#x8000	AnyModifier	
1	KEYCODE		key
	0	AnyKey	
1			pointer-mode
	0	Synchronous	
	1	Asynchronous	
1			keyboard-mode
	0	Synchronous	
	1	Asynchronous	
3			unused

UngrabKey

1	34		opcode
1	KEYCODE		key
	0	AnyKey	
2	3		request length
4	WINDOW		grab-window
2	SETofKEYMASK		modifiers
	#x8000	AnyModifier	
2			unused

AllowEvents

1	35		opcode
1			mode
	0	AsyncPointer	
	1	SyncPointer	
	2	ReplayPointer	
	3	AsyncKeyboard	
	4	SyncKeyboard	
	5	ReplayKeyboard	
	6	AsyncBoth	
	7	SyncBoth	
2	2		request length
4	TIMESTAMP		time
	0	CurrentTime	

GrabServer

1	36		opcode
1			unused
2	1		request length

UngrabServer

1	37		opcode
1			unused
2	1		request length

QueryPointer

1	38		opcode
1			unused
2	2		request length
4	WINDOW		window

=>

1	1		Reply
1	BOOL		same-screen
2	CARD16		sequence number
4	0		reply length
4	WINDOW		root
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		win-x
2	INT16		win-y

2	SETofKEYBUTMASK		mask
6			unused

GetMotionEvents

1	39		opcode
1			unused
2	4		request length
4	WINDOW		window
4	TIMESTAMP		start
	0	CurrentTime	
4	TIMESTAMP		stop
	0	CurrentTime	

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	2n		reply length
4	n		number of TIMECOORDs in events
20			unused
8n	LISTofTIMECOORD		events

TIMECOORD

4	TIMESTAMP		time
2	CARD16		x
2	CARD16		y

TranslateCoordinates

1	40		opcode
1			unused
2	4		request length
4	WINDOW		src-window
4	WINDOW		dst-window
2	INT16		src-x
2	INT16		src-y

=>

1	1		Reply
1	BOOL		same-screen
2	CARD16		sequence number
4	0		reply length
4	WINDOW		child
	0	None	
2	INT16		dst-x
2	INT16		dst-y
16			unused

WarpPointer

1	41		opcode
1			unused
2	6		request length
4	WINDOW		src-window
	0	None	
4	WINDOW		dst-window
	0	None	
2	INT16		src-x
2	INT16		src-y
2	CARD16		src-width
2	CARD16		src-height
2	INT16		dst-x
2	INT16		dst-y

SetInputFocus

1	42		opcode
1			revert-to
	0	None	

	1	PointerRoot	
	2	Parent	
2	3		request length
4	WINDOW		focus
	0	None	
	1	PointerRoot	
4	TIMESTAMP		time
	0	CurrentTime	

GetInputFocus

1	43		opcode
1			unused
2	1		request length

=>

1	1		Reply
1			revert-to
	0	None	
	1	PointerRoot	
	2	Parent	
2	CARD16		sequence number
4	0		reply length
4	WINDOW		focus
	0	None	
	1	PointerRoot	
20			unused

QueryKeymap

1	44		opcode
1			unused
2	1		request length

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	2		reply length
32	LISTofCARD8		keys

OpenFont

1	45		opcode
1			unused
2	3+(n+p)/4		request length
4	FONT		fid
2	n		length of name
2			unused
n	STRING8		name
p			unused, p=pad(n)

CloseFont

1	46		opcode
1			unused
2	2		request length
4	FONT		font

QueryFont

1	47		opcode
1			unused
2	2		request length
4	FONTABLE		font

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	7+2n+3m		reply length
12	CHARINFO		min-bounds

4			unused
12	CHARINFO		max-bounds
4			unused
2	CARD16		min-char-or-byte2
2	CARD16		max-char-or-byte2
2	CARD16		default-char
2	n		number of FONTPROPs in properties
1			draw-direction
	0	LeftToRight	
	1	RightToLeft	
1	CARD8		min-byte1
1	CARD8		max-byte1
1	BOOL		all-chars-exist
2	INT16		font-ascent
2	INT16		font-descent
4	m		number of CHARINFOs in char-infos
8n	LISTofFONTPROP		properties
12m	LISTofCHARINFO		char-infos

FONTPROP

4	ATOM	name
4	<32-bits>	value

CHARINFO

2	INT16	left-side-bearing
2	INT16	right-side-bearing
2	INT16	character-width
2	INT16	ascent
2	INT16	descent
2	CARD16	attributes

QueryTextExtents

1	48		opcode
1	BOOL		odd length, True if p = 2
2	2+(2n+p)/4		request length
4	FONTABLE		font
2n	STRING16		string
p			unused, p=pad(2n)

=>

1	1		Reply
1			draw-direction
	0	LeftToRight	
	1	RightToLeft	
2	CARD16		sequence number
4	0		reply length
2	INT16		font-ascent
2	INT16		font-descent
2	INT16		overall-ascent
2	INT16		overall-descent
4	INT32		overall-width
4	INT32		overall-left
4	INT32		overall-right
4			unused

ListFonts

1	49	opcode
1		unused
2	2+(n+p)/4	request length
2	CARD16	max-names
2	n	length of pattern
n	STRING8	pattern
p		unused, p=pad(n)

=>

1	1	Reply
1		unused

2	CARD16		sequence number
4	(n+p)/4		reply length
2	CARD16		number of STRs in names
22			unused
n	LISTofSTR		names
p			unused, p=pad(n)

ListFontsWithInfo

1	50		opcode
1			unused
2	2+(n+p)/4		request length
2	CARD16		max-names
2	n		length of pattern
n	STRING8		pattern
p			unused, p=pad(n)

=> (except for last in series)

1	1		Reply
1	n		length of name in bytes
2	CARD16		sequence number
4	7+2m+(n+p)/4		reply length
12	CHARINFO		min-bounds
4			unused
12	CHARINFO		max-bounds
4			unused
2	CARD16		min-char-or-byte2
2	CARD16		max-char-or-byte2
2	CARD16		default-char
2	m		number of FONTPROPs in properties
1			draw-direction
	0	LeftToRight	
	1	RightToLeft	
1	CARD8		min-byte1
1	CARD8		max-byte1
1	BOOL		all-chars-exist
2	INT16		font-ascent
2	INT16		font-descent
4	CARD32		replies-hint
8m	LISTofFONTPROP		properties
n	STRING8		name
p			unused, p=pad(n)

FONTPROP
encodings are the same as for QueryFont

CHARINFO
encodings are the same as for QueryFont

=> (last in series)

1	1	Reply
1	0	last-reply indicator
2	CARD16	sequence number
4	7	reply length
52		unused

SetFontPath

1	51	opcode
1		unused
2	2+(n+p)/4	request length
2	CARD16	number of STRs in path
2		unused
n	LISTofSTR	path
p		unused, p=pad(n)

GetFontPath

1	52	opcode
1		unused

```
  2    1                                  request list

=>
  1    1                                  Reply
  1                                       unused
  2    CARD16                             sequence number
  4    (n+p)/4                            reply length
  2    CARD16                             number of STRs in path
  22                                      unused
  n    LISTofSTR                          path
  p                                       unused, p=pad(n)

CreatePixmap
  1    53                                 opcode
  1    CARD8                              depth
  2    4                                  request length
  4    PIXMAP                             pid
  4    DRAWABLE                           drawable
  2    CARD16                             width
  2    CARD16                             height

FreePixmap
  1    54                                 opcode
  1                                       unused
  2    2                                  request length
  4    PIXMAP                             pixmap

CreateGC
  1    55                                 opcode
  1                                       unused
  2    4+n                                request length
  4    GCONTEXT                           cid
  4    DRAWABLE                           drawable
  4    BITMASK                            value-mask (has n 1-bits)
       #x00000001     function
       #x00000002     plane-mask
       #x00000004     foreground
       #x00000008     background
       #x00000010     line-width
       #x00000020     line-style
       #x00000040     cap-style
       #x00000080     join-style
       #x00000100     fill-style
       #x00000200     fill-rule
       #x00000400     tile
       #x00000800     stipple
       #x00001000     tile-stipple-x-origin
       #x00002000     tile-stipple-y-origin
       #x00004000     font
       #x00008000     subwindow-mode
       #x00010000     graphics-exposures
       #x00020000     clip-x-origin
       #x00040000     clip-y-origin
       #x00080000     clip-mask
       #x00100000     dash-offset
       #x00200000     dashes
       #x00400000     arc-mode
  4n   LISTofVALUE                        value-list
 VALUEs
  1                                       function
       0              Clear
       1              And
       2              AndReverse
       3              Copy
       4              AndInverted
```

	5	Noop	
	6	Xor	
	7	Or	
	8	Nor	
	9	Equiv	
	10	Invert	
	11	OrReverse	
	12	CopyInverted	
	13	OrInverted	
	14	Nand	
	15	Set	
4	CARD32		plane-mask
4	CARD32		foreground
4	CARD32		background
2	CARD16		line-width
1			line-style
	0	Solid	
	1	OnOffDash	
	2	DoubleDash	
1			cap-style
	0	NotLast	
	1	Butt	
	2	Round	
	3	Projecting	
1			join-style
	0	Miter	
	1	Round	
	2	Bevel	
1			fill-style
	0	Solid	
	1	Tiled	
	2	Stippled	
	3	OpaqueStippled	
1			fill-rule
	0	EvenOdd	
	1	Winding	
4	PIXMAP		tile
4	PIXMAP		stipple
2	INT16		tile-stipple-x-origin
2	INT16		tile-stipple-y-origin
4	FONT		font
1			subwindow-mode
	0	ClipByChildren	
	1	IncludeInferiors	
1	BOOL		graphics-exposures
2	INT16		clip-x-origin
2	INT16		clip-y-origin
4	PIXMAP		clip-mask
	0	None	
2	CARD16		dash-offset
1	CARD8		dashes
1			arc-mode
	0	Chord	
	1	PieSlice	

ChangeGC

1	56	opcode
1		unused
2	3+n	request length
4	GCONTEXT	gc
4	BITMASK	value-mask (has n 1-bits)
	encodings are the same as for CreateGC	
4n	LISTofVALUE	value-list
	encodings are the same as for CreateGC	

CopyGC

1	57	opcode
1		unused
2	4	request length
4	GCONTEXT	src-gc
4	GCONTEXT	dst-gc
4	BITMASK	value-mask

encodings are the same as for CreateGC

SetDashes

1	58	opcode
1		unused
2	3+(n+p)/4	request length
4	GCONTEXT	gc
2	CARD16	dash-offset
2	n	length of dashes
n	LISTofCARD8	dashes
p		unused, p=pad(n)

SetClipRectangles

1	59		opcode
1			ordering
	0	UnSorted	
	1	YSorted	
	2	YXSorted	
	3	YXBanded	
2	3+2n		request length
4	GCONTEXT		gc
2	INT16		clip-x-origin
2	INT16		clip-y-origin
8n	LISTofRECTANGLE		rectangles

FreeGC

1	60	opcode
1		unused
2	2	request length
4	GCONTEXT	gc

ClearArea

1	61	opcode
1	BOOL	exposures
2	4	request length
4	WINDOW	window
2	INT16	x
2	INT16	y
2	CARD16	width
2	CARD16	height

CopyArea

1	62	opcode
1		unused
2	7	request length
4	DRAWABLE	src-drawable
4	DRAWABLE	dst-drawable
4	GCONTEXT	gc
2	INT16	src-x
2	INT16	src-y
2	INT16	dst-x
2	INT16	dst-y
2	CARD16	width
2	CARD16	height

CopyPlane

1	63	opcode
1		unused
2	8	request length

4	DRAWABLE		src-drawable
4	DRAWABLE		dst-drawable
4	GCONTEXT		gc
2	INT16		src-x
2	INT16		src-y
2	INT16		dst-x
2	INT16		dst-y
2	CARD16		width
2	CARD16		height
4	CARD32		bit-plane

PolyPoint

1	64		opcode
1			coordinate-mode
	0	Origin	
	1	Previous	
2	3+n		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
4n	LISTofPOINT		points

PolyLine

1	65		opcode
1			coordinate-mode
	0	Origin	
	1	Previous	
2	3+n		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
4n	LISTofPOINT		points

PolySegment

1	66	opcode
1		unused
2	3+2n	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
8n	LISTofSEGMENT	segments

SEGMENT

2	INT16	x1
2	INT16	y1
2	INT16	x2
2	INT16	y2

PolyRectangle

1	67	opcode
1		unused
2	3+2n	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
8n	LISTofRECTANGLE	rectangles

PolyArc

1	68	opcode
1		unused
2	3+3n	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
12n	LISTofARC	arcs

FillPoly

1	69	opcode
1		unused
2	4+n	request length
4	DRAWABLE	drawable

4	GCONTEXT		gc
1			shape
	0	Complex	
	1	Nonconvex	
	2	Convex	
1			coordinate-mode
	0	Origin	
	1	Previous	
2			unused
4n	LISTofPOINT		points

PolyFillRectangle

1	70	opcode
1		unused
2	3+2n	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
8n	LISTofRECTANGLE	rectangles

PolyFillArc

1	71	opcode
1		unused
2	3+3n	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
12n	LISTofARC	arcs

PutImage

1	72		opcode
1			format
	0	Bitmap	
	1	XYPixmap	
	2	ZPixmap	
2	6+(n+p)/4		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
2	CARD16		width
2	CARD16		height
2	INT16		dst-x
2	INT16		dst-y
1	CARD8		left-pad
1	CARD8		depth
2			unused
n	LISTofBYTE		data
p			unused, p=pad(n)

GetImage

1	73		opcode
1			format
	1	XYPixmap	
	2	ZPixmap	
2	5		request length
4	DRAWABLE		drawable
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
4	CARD32		plane-mask

=>

1	1		Reply
1	CARD8		depth
2	CARD16		sequence number
4	(n+p)/4		reply length
4	VISUALID		visual
	0	None	

20		unused
n	LISTofBYTE	data
p		unused, p=pad(n)

PolyText8

1	74	opcode
1		unused
2	4+(n+p)/4	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
2	INT16	x
2	INT16	y
n	LISTofTEXTITEM8	items
p		unused, p=pad(n) (p is always 0 or 1)

TEXTITEM8

1	n	length of string (cannot be 255)
1	INT8	delta
n	STRING8	string
or		
1	255	font-shift indicator
1		font byte 3 (most significant)
1		font byte 2
1		font byte 1
1		font byte 0 (least significant)

PolyText16

1	75	opcode
1		unused
2	4+(n+p)/4	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
2	INT16	x
2	INT16	y
n	LISTofTEXTITEM16	items
p		unused, p=pad(n) (p is always 0 or 1)

TEXTITEM16

1	n	number of CHAR2Bs in string (cannot be 255)
1	INT8	delta
n	STRING16	string
or		
1	255	font-shift indicator
1		font byte 3 (most significant)
1		font byte 2
1		font byte 1
1		font byte 0 (least significant)

ImageText8

1	76	opcode
1	n	length of string
2	4+(n+p)/4	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
2	INT16	x
2	INT16	y
n	STRING8	string
p		unused, p=pad(n)

ImageText16

1	77	opcode
1	n	number of CHAR2Bs in string
2	4+(2n+p)/4	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
2	INT16	x

2	INT16		y
2n	STRING16		string
p			unused, p=pad(2n)

CreateColormap

1	78		opcode
1			alloc
	0	None	
	1	All	
2	4		request length
4	COLORMAP		mid
4	WINDOW		window
4	VISUALID		visual

FreeColormap

1	79	opcode
1		unused
2	2	request length
4	COLORMAP	cmap

CopyColormapAndFree

1	80	opcode
1		unused
2	3	request length
4	COLORMAP	mid
4	COLORMAP	src-cmap

InstallColormap

1	81	opcode
1		unused
2	2	request length
4	COLORMAP	cmap

UninstallColormap

1	82	opcode
1		unused
2	2	request length
4	COLORMAP	cmap

ListInstalledColormaps

1	83	opcode
1		unused
2	2	request length
4	WINDOW	window

=>

1	1	Reply
1		unused
2	CARD16	sequence number
4	n	reply length
2	n	number of COLORMAPs in cmaps
22		unused
4n	LISTofCOLORMAP	cmaps

AllocColor

1	84	opcode
1		unused
2	4	request length
4	COLORMAP	cmap
2	CARD16	red
2	CARD16	green
2	CARD16	blue
2		unused

=>

1	1	Reply

1		unused
2	CARD16	sequence number
4	0	reply length
2	CARD16	red
2	CARD16	green
2	CARD16	blue
2		unused
4	CARD32	pixel
12		unused

AllocNamedColor

1	85	opcode
1		unused
2	3+(n+p)/4	request length
4	COLORMAP	cmap
2	n	length of name
2		unused
n	STRING8	name
p		unused, p=pad(n)

=>

1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length
4	CARD32	pixel
2	CARD16	exact-red
2	CARD16	exact-green
2	CARD16	exact-blue
2	CARD16	visual-red
2	CARD16	visual-green
2	CARD16	visual-blue
8		unused

AllocColorCells

1	86	opcode
1	BOOL	contiguous
2	3	request length
4	COLORMAP	cmap
2	CARD16	colors
2	CARD16	planes

=>

1	1	Reply
1		unused
2	CARD16	sequence number
4	n+m	reply length
2	n	number of CARD32s in pixels
2	m	number of CARD32s in masks
20		unused
4n	LISTofCARD32	pixels
4m	LISTofCARD32	masks

AllocColorPlanes

1	87	opcode
1	BOOL	contiguous
2	4	request length
4	COLORMAP	cmap
2	CARD16	colors
2	CARD16	reds
2	CARD16	greens
2	CARD16	blues

=>

1	1	Reply
1		unused
2	CARD16	sequence number

4	n	reply length
2	n	number of CARD32s in pixels
2		unused
4	CARD32	red-mask
4	CARD32	green-mask
4	CARD32	blue-mask
8		unused
4n	LISTofCARD32	pixels

FreeColors

1	88	opcode
1		unused
2	3+n	request length
4	COLORMAP	cmap
4	CARD32	plane-mask
4n	LISTofCARD32	pixels

StoreColors

1	89	opcode
1		unused
2	2+3n	request length
4	COLORMAP	cmap
12n	LISTofCOLORITEM	items

COLORITEM

4	CARD32	pixel
2	CARD16	red
2	CARD16	green
2	CARD16	blue
1		do-red, do-green, do-blue
	#x01	do-red (1 is True, 0 is False)
	#x02	do-green (1 is True, 0 is False)
	#x04	do-blue (1 is True, 0 is False)
	#xf8	unused
1		unused

StoreNamedColor

1	90	opcode
1		do-red, do-green, do-blue
	#x01	do-red (1 is True, 0 is False)
	#x02	do-green (1 is True, 0 is False)
	#x04	do-blue (1 is True, 0 is False)
	#xf8	unused
2	4+(n+p)/4	request length
4	COLORMAP	cmap
4	CARD32	pixel
2	n	length of name
2		unused
n	STRING8	name
p		unused, p=pad(n)

QueryColors

1	91	opcode
1		unused
2	2+n	request length
4	COLORMAP	cmap
4n	LISTofCARD32	pixels

=>

1	1	Reply
1		unused
2	CARD16	sequence number
4	2n	reply length
2	n	number of RGBs in colors
22		unused
8n	LISTofRGB	colors

RGB

2	CARD16		red
2	CARD16		green
2	CARD16		blue
2			unused

LookupColor

1	92		opcode
1			unused
2	3+(n+p)/4		request length
4	COLORMAP		cmap
2	n		length of name
2			unused
n	STRING8		name
p			unused, p=pad(n)

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
2	CARD16		exact-red
2	CARD16		exact-green
2	CARD16		exact-blue
2	CARD16		visual-red
2	CARD16		visual-green
2	CARD16		visual-blue
12			unused

CreateCursor

1	93		opcode
1			unused
2	8		request length
4	CURSOR		cid
4	PIXMAP		source
4	PIXMAP		mask
	0	None	
2	CARD16		fore-red
2	CARD16		fore-green
2	CARD16		fore-blue
2	CARD16		back-red
2	CARD16		back-green
2	CARD16		back-blue
2	CARD16		x
2	CARD16		y

CreateGlyphCursor

1	94		CreateGlyphCursor
1			unused
2	8		request length
4	CURSOR		cid
4	FONT		source-font
4	FONT		mask-font
	0	None	
2	CARD16		source-char
2	CARD16		mask-char
2	CARD16		fore-red
2	CARD16		fore-green
2	CARD16		fore-blue
2	CARD16		back-red
2	CARD16		back-green
2	CARD16		back-blue

FreeCursor

1	95		opcode
1			unused

2	2		request length
4	CURSOR		cursor

RecolorCursor

1	96		opcode
1			unused
2	5		request length
4	CURSOR		cursor
2	CARD16		fore-red
2	CARD16		fore-green
2	CARD16		fore-blue
2	CARD16		back-red
2	CARD16		back-green
2	CARD16		back-blue

QueryBestSize

1	97		opcode
1			class
	0	Cursor	
	1	Tile	
	2	Stipple	
2	3		request length
4	DRAWABLE		drawable
2	CARD16		width
2	CARD16		height

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
2	CARD16		width
2	CARD16		height
20			unused

QueryExtension

1	98		opcode
1			unused
2	2+(n+p)/4		request length
2	n		length of name
2			unused
n	STRING8		name
p			unused, p=pad(n)

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
1	BOOL		present
1	CARD8		major-opcode
1	CARD8		first-event
1	CARD8		first-error
20			unused

ListExtensions

1	99		opcode
1			unused
2	1		request length

=>

1	1		Reply
1	CARD8		number of STRs in names
2	CARD16		sequence number
4	(n+p)/4		reply length
24			unused
n	LISTofSTR		names

p			unused, p=pad(n)

ChangeKeyboardMapping

1	100		opcode
1	n		keycode-count
2	2+nm		request length
1	KEYCODE		first-keycode
1	m		keysyms-per-keycode
2			unused
4nm	LISTofKEYSYM		keysyms

GetKeyboardMapping

1	101		opcode
1			unused
2	2		request length
1	KEYCODE		first-keycode
1	CARD8		count
2			unused

=>

1	1		Reply
1	n		keysyms-per-keycode
2	CARD16		sequence number
4	nm		reply length (m = count field from the request)
24			unused
4nm	LISTofKEYSYM	keysyms	

ChangeKeyboardControl

1	102		opcode
1			unused
2	2+n		request length
4	BITMASK		value-mask (has n 1-bits)
	#x0001	key-click-percent	
	#x0002	bell-percent	
	#x0004	bell-pitch	
	#x0008	bell-duration	
	#x0010	led	
	#x0020	led-mode	
	#x0040	key	
	#x0080	auto-repeat-mode	
4n	LISTofVALUE		value-list

VALUEs

1	INT8		key-click-percent
1	INT8		bell-percent
2	INT16		bell-pitch
2	INT16		bell-duration
1	CARD8		led
1			led-mode
	0	Off	
	1	On	
1	KEYCODE		key
1			auto-repeat-mode
	0	Off	
	1	On	
	2	Default	

GetKeyboardControl

1	103		opcode
1			unused
2	1		request length

=>

1	1		Reply
1			global-auto-repeat
	0	Off	
	1	On	

2	CARD16		sequence number
4	5		reply length
4	CARD32		led-mask
1	CARD8		key-click-percent
1	CARD8		bell-percent
2	CARD16		bell-pitch
2	CARD16		bell-duration
2			unused
32	LISTofCARD8	auto-repeats	

Bell

1	104		opcode
1	INT8		percent
2	1		request length

ChangePointerControl

1	105		opcode
1			unused
2	3		request length
2	INT16		acceleration-numerator
2	INT16		acceleration-denominator
2	INT16		threshold
1	BOOL		do-acceleration
1	BOOL		do-threshold

GetPointerControl

1	106		GetPointerControl
1			unused
2	1		request length

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
2	CARD16		acceleration-numerator
2	CARD16		acceleration-denominator
2	CARD16		threshold
18			unused

SetScreenSaver

1	107		opcode
1			unused
2	3		request length
2	INT16		timeout
2	INT16		interval
1			prefer-blanking
	0	No	
	1	Yes	
	2	Default	
1			allow-exposures
	0	No	
	1	Yes	
	2	Default	
2			unused

GetScreenSaver

1	108		opcode
1			unused
2	1		request length

=>

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length

2	CARD16		timeout
2	CARD16		interval
1			prefer-blanking
	0	No	
	1	Yes	
1			allow-exposures
	0	No	
	1	Yes	
18			unused

ChangeHosts

1	109		opcode
1			mode
	0	Insert	
	1	Delete	
2	2+(n+p)/4		request length
1			family
	0	Internet	
	1	DECnet	
	2	Chaos	
1			unused
2	CARD16		length of address
n	LISTofCARD8		address
p			unused, p=pad(n)

ListHosts

1	110		opcode
1			unused
2	1		request length

=>

1	1		Reply
1			mode
	0	Disabled	
	1	Enabled	
2	CARD16		sequence number
4	n/4		reply length
2	CARD16		number of HOSTs in hosts
22			unused
n	LISTofHOST		hosts (n always a multiple of 4)

SetAccessControl

1	111		opcode
1			mode
	0	Disable	
	1	Enable	
2	1		request length

SetCloseDownMode

1	112		opcode
1			mode
	0	Destroy	
	1	RetainPermanent	
	2	RetainTemporary	
2	1		request length

KillClient

1	113		opcode
1			unused
2	2		request length
4	CARD32		resource
	0	AllTemporary	

RotateProperties

1	114		opcode
1			unused

2	3+n		request length
4	WINDOW		window
2	n		number of properties
2	INT16		delta
4n	LISTofATOM		properties

ForceScreenSaver

1	115		ForceScreenSaver
1			mode
	0	Reset	
	1	Activate	
2	1		request length

SetPointerMapping

1	116		opcode
1	n		length of map
2	1+(n+p)/4		request length
n	LISTofCARD8		map
p			unused, p=pad(n)

=>

1	1		Reply
1			status
	0	Success	
	1	Busy	
2	CARD16		sequence number
4	0		reply length
24			unused

GetPointerMapping

1	117		opcode
1			unused
2	1		request length

=>

1	1		Reply
1	n		length of map
2	CARD16		sequence number
4	(n+p)/4		reply length
24			unused
n	LISTofCARD8		map
p			unused, p=pad(n)

SetModifierMapping

1	118		opcode
1	n		keycodes-per-modifier
2	1+2n		request length
8n	LISTofKEYCODE		keycodes

=>

1	1		Reply
1			status
	0	Success	
	1	Busy	
	2	Failed	
2	CARD16		sequence number
4	0		reply length
24			unused

GetModifierMapping

1	119		opcode
1			unused
2	1		request length

=>

1	1		Reply
1	n		keycodes-per-modifier

2	CARD16		sequence number
4	2n		reply length
24			unused
8n	LISTofKEYCODE		keycodes

NoOperation

1	127		opcode
1			unused
2	1		request length

Events

KeyPress

1	2		code
1	KEYCODE		detail
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

KeyRelease

1	3		code
1	KEYCODE		detail
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

ButtonPress

1	4		code
1	BUTTON		detail
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

ButtonRelease

1	5		code
1	BUTTON		detail

2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

MotionNotify

1	6		code
1			detail
	0	Normal	
	1	Hint	
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

EnterNotify

1	7		code
1			detail
	0	Ancestor	
	1	Virtual	
	2	Inferior	
	3	Nonlinear	
	4	NonlinearVirtual	
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1			mode
	0	Normal	
	1	Grab	
	2	Ungrab	
1			same-screen, focus
	#x01	focus (1 is True, 0 is False)	
	#x02	same-screen (1 is True, 0 is False)	
	#xfc	unused	

LeaveNotify

1	8		code
1			detail
	0	Ancestor	

	1	Virtual	
	2	Inferior	
	3	Nonlinear	
	4	NonlinearVirtual	
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1			mode
	0	Normal	
	1	Grab	
	2	Ungrab	
1			same-screen, focus
	#x01	focus (1 is True, 0 is False)	
	#x02	same-screen (1 is True, 0 is False)	
	#xfc	unused	

FocusIn

1	9		code
1			detail
	0	Ancestor	
	1	Virtual	
	2	Inferior	
	3	Nonlinear	
	4	NonlinearVirtual	
	5	Pointer	
	6	PointerRoot	
	7	None	
2	CARD16		sequence number
4	WINDOW		event
1			mode
	0	Normal	
	1	Grab	
	2	Ungrab	
	3	WhileGrabbed	
23			unused

FocusOut

1	10		code
1			detail
	0	Ancestor	
	1	Virtual	
	2	Inferior	
	3	Nonlinear	
	4	NonlinearVirtual	
	5	Pointer	
	6	PointerRoot	
	7	None	
2	CARD16		sequence number
4	WINDOW		event
1			mode
	0	Normal	
	1	Grab	
	2	Ungrab	
	3	WhileGrabbed	
23			unused

KeymapNotify

1	11		code
31	LISTofCARD8		keys (byte for keycodes 0-7 is omitted)

Expose

1	12		code
1			unused
2	CARD16		sequence number
4	WINDOW		window
2	CARD16		x
2	CARD16		y
2	CARD16		width
2	CARD16		height
2	CARD16		count
14			unused

GraphicsExposure

1	13		code
1			unused
2	CARD16		sequence number
4	DRAWABLE		drawable
2	CARD16		x
2	CARD16		y
2	CARD16		width
2	CARD16		height
2	CARD16		minor-opcode
2	CARD16		count
1	CARD8		major-opcode
11			unused

NoExposure

1	14		code
1			unused
2	CARD16		sequence number
4	DRAWABLE		drawable
2	CARD16		minor-opcode
1	CARD8		major-opcode
21			unused

VisibilityNotify

1	15		code
1			unused
2	CARD16		sequence number
4	WINDOW		window
1			state
	0	Unobscured	
	1	PartiallyObscured	
	2	FullyObscured	
23			unused

CreateNotify

1	16		code
1			unused
2	CARD16		sequence number
4	WINDOW		parent
4	WINDOW		window
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
1	BOOL		override-redirect
9			unused

DestroyNotify

1	17		code

1			unused
2	CARD16		sequence number
4	WINDOW		event
4	WINDOW		window
20			unused

UnmapNotify

1	18		code
1			unused
2	CARD16		sequence number
4	WINDOW		event
4	WINDOW		window
1	BOOL		from-configure
19			unused

MapNotify

1	19		code
1			unused
2	CARD16		sequence number
4	WINDOW		event
4	WINDOW		window
1	BOOL		override-redirect
19			unused

MapRequest

1	20		code
1			unused
2	CARD16		sequence number
4	WINDOW		parent
4	WINDOW		window
20			unused

ReparentNotify

1	21		code
1			unused
2	CARD16		sequence number
4	WINDOW		event
4	WINDOW		window
4	WINDOW		parent
2	INT16		x
2	INT16		y
1	BOOL		override-redirect
11			unused

ConfigureNotify

1	22		code
1			unused
2	CARD16		sequence number
4	WINDOW		event
4	WINDOW		window
4	WINDOW		above-sibling
	0	None	
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
1	BOOL		override-redirect
5			unused

ConfigureRequest

1	23		code
1			stack-mode
	0	Above	
	1	Below	
	2	TopIf	

	3	BottomIf	
	4	Opposite	
2	CARD16		sequence number
4	WINDOW		parent
4	WINDOW		window
4	WINDOW		sibling
	0	None	
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
2	BITMASK		value-mask
	#x0001	x	
	#x0002	y	
	#x0004	width	
	#x0008	height	
	#x0010	border-width	
	#x0020	sibling	
	#x0040	stack-mode	
4			unused

GravityNotify

1	24	code
1		unused
2	CARD16	sequence number
4	WINDOW	event
4	WINDOW	window
2	INT16	x
2	INT16	y
16		unused

ResizeRequest

1	25	code
1		unused
2	CARD16	sequence number
4	WINDOW	window
2	CARD16	width
2	CARD16	height
20		unused

CirculateNotify

1	26		code
1			unused
2	CARD16		sequence number
4	WINDOW		event
4	WINDOW		window
4	WINDOW		parent
1			place
	0	Top	
	1	Bottom	
15			unused

CirculateRequest

1	27		code
1			unused
2	CARD16		sequence number
4	WINDOW		parent
4	WINDOW		window
4			unused
1			place
	0	Top	
	1	Bottom	
15			unused

PropertyNotify

1	28		code
1			unused
2	CARD16		sequence number
4	WINDOW		window
4	ATOM		atom
4	TIMESTAMP		time
1			state
	0	NewValue	
	1	Deleted	
15			unused

SelectionClear

1	29		code
1			unused
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		owner
4	ATOM		selection
16			unused

SelectionRequest

1	30		code
1			unused
2	CARD16		sequence number
4	TIMESTAMP		time
	0	CurrentTime	
4	WINDOW		owner
4	WINDOW		requestor
4	ATOM		selection
4	ATOM		target
4	ATOM		property
	0	None	
4			unused

SelectionNotify

1	31		code
1			unused
2	CARD16		sequence number
4	TIMESTAMP		time
	0	CurrentTime	
4	WINDOW		requestor
4	ATOM		selection
4	ATOM		target
4	ATOM		property
	0	None	
8			unused

ColormapNotify

1	32		code
1			unused
2	CARD16		sequence number
4	WINDOW		window
4	COLORMAP		colormap
	0	None	
1	BOOL		new
1			state
	0	Uninstalled	
	1	Installed	
18			unused

ClientMessage

1	33		code
1	CARD8		format
2	CARD16		sequence number
4	WINDOW		window

```
     4     ATOM                              type
    20                                       data

MappingNotify
     1     34                                code
     1                                       unused
     2     CARD16                            sequence number
     1                                       request
           0              Modifier
           1              Keyboard
           2              Pointer
     1     KEYCODE                           first-keycode
     1     CARD8                             count
    25                                       unused
```

Index

Xlib – C Language X Interface

X Window System

X Version 11, Release 2

Jim Gettys

Digital Equipment Corporation
Systems Research Center
MIT Project Athena

Ron Newman

Massachusetts Institute of Technology
MIT Project Athena

Robert W. Scheifler

Massachusetts Institute of Technology
Laboratory for Computer Science

Acknowledgments

Whereas the design and implementation of the first 10 versions of X were primarily the work of three individuals, Robert Scheifler of the MIT Laboratory for Computer Science and Jim Gettys of Digital Equipment Corporation and Ron Newman of MIT, both at MIT Project Athena, X version 11 is the result of the efforts of literally dozens of individuals at almost as many locations and organizations. At the risk of offending some of the players by exclusion, many people deserve special credit and recognition. Our apologies to anyone inadvertently overlooked.

First, Phil Karlton and Scott McGregor, both of Digital, for their considerable contributions to the specification of the version 11 protocol. Sue Angebranndt, Raymond Drewry, Todd Newman and Phil Karlton of Digital have worked long and hard to produce the sample server implementation.

Todd Brunhoff of Tektronix was "loaned" to Project Athena at exactly the right moment to provide very capable and much needed assistance during the alpha and beta releases. He is totally responsible for the successfull integration of sources from multiple sites; we simply wouldn't have a release without him.

Ralph Swick of Project Athena and Digital kept it all together for us. He has handled literally thousands of requests for people everywhere and saved the sanity of one of us (JG). His calm good cheer has been a foundation on which we could build.

Special thanks must also go to Sam Fuller, Vice President of Corporate Research at Digital, who has remained committed to the widest public availability of X and who made it possible to greatly supplement MIT's resources with the Digital staff in order to make version 11 a reality. Many of the people mentioned here are part of the Western Software Laboratory (Digital UEG-WSL) of the ULTRIX Engineering group, who work for Smokey Wallace, who has been vital to the project's success; additional others have worked on the toolkit, and are acknowledged there.

Our thanks also go to Al Mento and Al Wojtas of Digital's ULTRIX Documentation Group. With good humor and cheer, they took a rough draft of the V11 Xlib manual and made it an infinitely better and more useful document. The work they have done will help many everywhere. We also would like to thank Hal Murray (Digital SRC) and Peter George (Digital VMS) who contributed much by proof reading this manual.

We would like to thank Jeff Dike (Digital UEG), Tom Benson, Jackie Granfield, and Vince Orgovan (Digital VMS) who helped with the library utilities implementation.

Hania Gajewska (Digital UEG-WSL) was instrumental in the semantic design of the window manager properties along with Ellis Cohen of CMU and Siemens.

Dave Rosenthal of Sun Microsystems also contributed to the protocol and provided the sample generic color frame buffer device dependent code.

The alpha and beta test participants deserve recognition as well. It is significant that the bug reports (and many fixes) during alpha and beta test came almost exclusively from just a few of the alpha testers, mostly hardware vendors working on product implementations of X. The continued public contribution of vendors and universities is certainly to the benefit of the entire X community.

The Version 11 protocol was authored by Bob Scheifler of MIT's Laboratory for computer science. Contributers to the design were:

Dave Carver (Digital HPW)
Branko Gerovac (Digital HPW)
Jim Gettys (MIT/Project Athena, Digital)
Phil Karlton (Digital WSL)
Scott McGregor (Digital SSG)
Ram Rao (Digital UEG)
David Rosenthal (Sun)
Dave Winchell (Digital UEG)

Invited reviewers who provided useful input:

Andrew Cherenson (Berkeley)
Burns Fisher (Digital)
Dan Garfinkel (HP)
Leo Hourvitz (Next)
Brock Krizan (HP)
David Laidlaw (Stellar)
Dave Mellinger (Interleaf)
Ron Newman (MIT)
John Ousterhout (Berkeley)
Andrew Palay (ITC CMU)
Ralph Swick (MIT/Project Athena, Digital)
Craig Taylor (Sun)
Jeffery Vroom (Stellar)

And of course, we must acknowledge Paul Asente, of Stanford University, who wrote W, the predecessor to X.

And thanks must also go to MIT, Digital Equipment Corporation, and IBM for providing the environment where it could happen.

Jim Gettys
Systems Research Center
MIT / Project Athena
Digital Equipment Corporation

Ron Newman
Project Athena
Massachusetts Institute of Technology

Robert W. Scheifler
Laboratory for Computer Science
Massachusetts Institute of Technology

September 15, 1987

Chapter 1

Introduction to Xlib

The X Window System is a network transparent window system that was designed at MIT and that runs under 4.3BSD UNIX, ULTRIX-32, many other UNIX variants, VAX/VMS, MS/DOS, as well as several other operating systems.

X display servers run on computers with either monochrome or color bitmap display hardware. The server distributes user input to and accepts output requests from various client programs located either on the same machine or elsewhere in your network. Xlib is a C subroutine library that application programs (clients) use to interface with the window system by means of a stream connection. Although a client usually runs on the same machine as the X server it is talking to, this need not be the case.

This manual is a reference guide to the low-level C language interface to the X Window System protocol. It is neither a tutorial nor a user guide to programming to the X Window System. Other high-level abstractions (for example, those provided by the toolkits for X) are built on top of the Xlib library. For further information about these high-level libraries, see the appropriate toolkit documentation. For general information about the design of X, see ''The X Window System.'' The *X Window System Protocol* provides the definitive word on the behavior of X. While additional information is provided by this manual, the protocol document is the ruling document.

This manual assumes a basic understanding of a graphics window system and of the C programming language. To provide an introduction to X programming, this Chapter discusses:

- Overview of the X Window System
- Naming and argument conventions
- Programming considerations
- Documentation conventions

1.1. Overview of the X Window System

Some of the terms used in this book are unique to X, while other terms that are common to other window systems have different meanings in X. You may find it helpful to refer to the glossary, located at the end of the book, when you are uncertain of a term's meaning in the context of the X Window System.

The X Window System supports one or more screens containing overlapping windows or subwindows. A screen is a physical monitor and hardware, which may be either color or black and white. There can be multiple screens per display or workstation. A single server can provide display services for any number of screens. A set of screens for a single user with one keyboard and one mouse is called a ''display''.

All the windows in an X server are arranged in a strict hierarchy. At the top of the hierarchy are the root windows, which cover each of the display screens. Each root window is partially or completely covered by child windows. All windows, except for root windows, have parents. There is usually at least one window per application program. Child windows may in turn have their own children. In this way, an application program can create a tree of arbitrary depth on each screen.

A child window may be larger than its parent. That is, part or all of the child window may extend beyond the boundaries of the parent. However, all output to a window is clipped by the boundaries of its parent window. If several children of a window have overlapping locations, one of the children is considered to be on top of or raised over the others, obscuring them. Output to areas covered by other windows will be suppressed by the window system. If a window is obscured by a second window, the second window will obscure only those ancestors of the second window, which are also ancestors of the first window.

A window has a border of zero or more pixels in width, which can be any pattern (pixmap) or solid color you like. A window usually but not always has a background pattern which will be repainted by the window system when uncovered. Each window has its own coordinate system. Child windows obscure their parents unless the child windows have no background, and graphic operations in the parent window usually are clipped by the children.

Input from X takes the form of events. Events may either be side effects of a command (for example, restacking windows generates exposure events) or completely asynchronous (for example, the keyboard). A client program asks to be informed of events. Programs must be prepared to handle (or ignore) events of all types, because other applications can send events to your application.

X does not take responsibility for the contents of windows. When part or all of a window is hidden and then brought back onto the screen, its contents may be lost, and the client program is notified (by an exposure event) that part or all of the window needs to be repainted. Programs must be prepared to regenerate the contents of windows on demand.

X also provides off screen storage of graphics objects, called "pixmaps." Single plane (depth 1) pixmaps are sometimes referred to as "bitmaps." These can be used in most graphics functions interchangeably with windows, and are used in various graphics operations to define patterns also called "tiles." Windows and pixmaps together are referred to as "drawables."

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

If it does not want a request to execute asynchronously, a client can follow it with a call to XSync, which will block until all previously buffered asynchronous events have been sent and acted on. As an important side effect, the output buffer is always flushed by a call to any function which returns a value from the server or waits for input (for example, XPending, XNextEvent, XWindowEvent, XFlush, or XSync).

Many Xlib functions will return an integer resource ID, which allows you to refer to objects stored on the X server. These can be of type Window, Font, Pixmap, Bitmap, Cursor, and GContext, as defined in the file <X11/X.h>. These resources are created by user requests, and destroyed (or freed) by user requests or when connections are closed. Most of these resources are potentially sharable between applications, and in fact, windows are manipulated explicitly by window manager programs. Fonts and cursors are typically shared automatically since the X server treats fonts specially, loading and unloading font storage as needed. GContexts should not be shared between applications.

Some functions return Status, an integer error indication. If the function fails, it will return a zero. If the function returns a status of zero, it has not updated the return parameters. Because C does not provide multiple return values, many functions must return their results by writing into client-passed storage. Any pointer to a structure that is used to return a value is designated as such by the *_return* suffix as part of its name. All other pointers passed to these functions are used for reading only. A few arguments use pointers to structures that are used for both input and output and are indicated by using the *_in_out* suffix. By default, errors are handled either by a standard library function or by one that you provide. Functions that return pointers to strings will return NULL pointers if the string does not exist.

Input events (for example, key pressed or mouse moved) arrive asynchronously from the server and are queued until they are requested by an explicit call (for example, XNextEvent or

The <> has the meaning defined by the # include statement of the C compiler and is a file relative to a well known directory. On UNIX-based systems, this is /usr/include.

XWindowEvent). In addition, some of the library functions (for example, XResizeWindow and XRaiseWindow) generate exposure events (that is, requests to repaint Sections of a window that do not have valid contents). These events also arrive asynchronously, but the client may wish to explicitly wait for them by calling XSync after calling a function which may generate exposure events.

1.2. Naming and Argument Conventions within Xlib

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

The major naming conventions are:

- To better differentiate the X symbols from the user symbols, the library uses mixed case for external symbols, and leaves lower case for variables and all upper case for user macros, per existing convention.
- All Xlib functions begin with a capital X.
- The beginnings of all procedure names and symbols are capitalized.
- All user-visible data structures begin with a capital X. More generally, anything that a user might dereference begins with an capital X.
- Macros and other symbols do not begin with a capital X. To distinguish them from all user symbols, each word in the macro is capitalized.
- All elements of or variables in a data structure are in lower case. Compound words, where needed, are constructed with underscores (_).
- The display argument, where used, is always first in the argument list.
- All resource objects, where used, occur at the beginning of the argument list, immediately after the display variable.
- When a graphics context is present together with another type of resource (most commonly, a drawable), the graphics context occurs in the argument list after the other resource. Drawables outrank all other resources.
- Source arguments always precede the destination arguments in the argument list.
- The x argument always precedes the y argument in the argument list.
- The width argument always precedes the height argument in the argument list.
- Where the x, y, width and height arguments are used together, the x and y arguments always precede the width and height arguments.
- Where an array occurs in an argument list accompanied with a count (number of elements in the array), the array always precedes the count in the argument list.
- Where a mask is accompanied with a structure, the mask always precedes the pointer to the structure in the argument list.

1.3. Programming Considerations

The major considerations are:

- Keyboards are the greatest variable between different manufacturer's workstations. If you want your program to be portable, you should be particularly conservative here.
- Many display systems have limited amounts of off-screen memory. If you can, you should minimize use of pixmaps and backing store.
- The user should have control of his screen real-estate. Therefore, you should write your applications to react to window management, rather than presume control of the entire screen. What you do inside of your top-level window, however, is up to your application. There is more on this topic elsewhere in the book.

- Coordinates and sizes in X are actually 16-bit quantities. They usually are declared as an ''int'' in the interface (int is 16-bits on some machines). Values larger than 16 bits are truncated silently. Sizes (width and height) are unsigned quantities. This decision was taken to minimize the bandwidth required for a given level of performance.

1.4. Conventions Used in this Manual

The major documentation conventions are:

- Global symbols in this manual are printed in this special font. These can be either procedure names, symbols defined in include files, or structure names. Arguments, user-supplied variables, are printed in *italics*.
- Most procedures are introduced by a general discussion that may be used to distinguish this procedure from other procedures and are followed by the procedure declaration itself. Each argument is then specifically explained. General discussion of the procedure, if any is required, follows the arguments. Where applicable, the last paragraph of given explanation lists the possible Xlib error codes that can be generated by that function. See Section 8.10.2 for a complete discussion of the Xlib error codes.
- To eliminate any ambiguity between those arguments that you pass and those that a function returns to you, the explanations for all arguments that you pass start with the word ''specifies''. By contrast, the explanations for all arguments that are returned to you start with the word ''returns''.

Chapter 2

Display Functions

Before your program can use a display, you must establish a connection to the X server driving your display. Once you have established a connection, you then can use the Xlib macros and functions discussed in this Chapter to return information about the display. This Chapter discusses how to:

- Open (connect) the display
- Obtain information about the display, image formats, or a screen
- Generate a NoOperation protocol request
- Free client-created data
- Close (disconnect) a display

Finally, the Chapter concludes with a Section that describes the operations that occur when the connection to the X server is closed.

2.1. Opening the Display

To open a connection to the X server controlling the specified display, use XOpenDisplay.

Display *XOpenDisplay(*display_name*)
    char **display_name*;

display_name	Specifies the hardware display name, which determines the display and communications domain to be used. On a UNIX-based system, if the display_name is NULL, it defaults to the value of the DISPLAY environment variable. On all non-UNIX systems, see the Xlib manual associated with your operating system to determine the default for this argument.

On UNIX-based systems, the display name or DISPLAY environment variable is a string that has the format:

hostname:*number*.*screen_number*

hostname	Specifies the name of the host machine on which the display is physically attached. You follow the hostname with either a single colon (:) or a double colon (::). Each is discussed in the following paragraph.
number	Specifies the number of the display server on that host machine. You may optionally follow this display number with a period (.).
screen_number	Specifies which screen should be used on that host server. Multiple screens can be connected to or controlled by a single X server. The screen_number sets an internal variable that can be accessed by using the DefaultScreen macro (or the XDefaultScreen function if you are using other languages). See Section 2.2.1 for further information.

For example, the following would specifies screen 2 of display 0 on the machine named mit-athena:

mit-athena:0.2

The XOpenDisplay function returns a Display structure that serves as the connection to the X server and that contains all the information about that X server. XOpenDisplay connects the specified hardware display to the server through TCP, UNIX domain, or DECnet stream communications protocols. If the hostname is a host machine name and a single colon (:) separates the hostname and display number, XOpenDisplay connects using TCP streams. If the hostname is *unix* and a single colon (:) separates it from the display number, XOpenDisplay connects using UNIX domain IPC streams. If the hostname is a host machine name and a double colon (::) separates the hostname and display number, XOpenDisplay connects using DECnet streams. To use DECnet, however, the X library implementation must have been built to support it. A single server can support any or all of these transport mechanisms simultaneously.

If successful, XOpenDisplay returns a pointer to a Display structure that is defined in <X11/Xlib.h>. See Section 2.2.1, for information about using macros and functions to obtain information from the Display structure. If XOpenDisplay does not succeed, it returns a NULL. After a successful call to XOpenDisplay, all of the screens in the display may be used by the client application. The screen number specified in the display_name argument will be returned by the DefaultScreen macro or the XDefaultScreen function. You can access elements of the Display and Screen structures by using the information macros or functions.

X servers may implement various types of access control mechanisms. See Section 7.11 for information about access control.

2.2. Obtaining Information About the Display, Image Formats, or Screens

The Xlib library provides a number of useful macros and corresponding functions that return data from the Display structure. The macros are to be used for C programming, while their corresponding function equivalents are for other language bindings. This Section discusses the:

- Display macros
- Image format macros
- Screen macros

All other members of the Display structure (that is, those for which no macros are defined) are private to Xlib and must not be used. That is, applications must never directly modify or inspect these private elements in the Display structure.

Note

> Some of the functions in the following Sections are misnamed (for example, XDisplayWidth). These functions really should be named Screen*whatever* and XScreen*whatever*, not Display*whatever* or XDisplay*whatever*. Our apologies for the resulting confusion.

2.2.1. Display Macros

Applications should not directly modify any part of the Display and Screen structures. The members should be considered read-only, although they may change as the result of other operations on the display.

The following lists the C language macros, their corresponding function equivalents that are for other language bindings, and what data they both can return.

AllPlanes()

unsigned long XAllPlanes()

Both return a value with all bits set on suitable for use in a plane argument to a procedure.

Both BlackPixel and WhitePixel can be used in implementing a ''monochrome'' application. These pixel values are for permanently allocated entries in the default colormap. The actual RGB values are settable on some screens and, in any case, may not actually be ''black'' or ''white''.

The names are intended to convey the expected relative intensity of the colors.

BlackPixel(*display*, *screen_number*)

unsigned long XBlackPixel(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the black pixel value for the specified screen.

WhitePixel(*display*, *screen_number*)

unsigned long XWhitePixel(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the white pixel value for the specified screen.

ConnectionNumber(*display*)

int XConnectionNumber(*display*)
    Display **display*;

Both return a connection number for the specified display. On a UNIX-based system, this is the file descriptor of the connection.

DefaultColormap(*display*, *screen_number*)

Colormap XDefaultColormap(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the default Colormap ID for allocation on the specified screen. Most routine allocations of color should be made out of this colormap.

DefaultDepth(*display*, *screen_number*)

int XDefaultDepth(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the depth (number of planes) of the default root window for the specified screen. Other depths may also be supported on this screen. See XMatchVisualInfo in Chapter 10 to find out how to determine what depths may be available.

DefaultGC(*display*, *screen_number*)

GC XDefaultGC(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the default graphics context for the root window of the specified screen. This GC is created for the convenience of simple applications and contains the default GC components with the foreground and background pixel values initialized to the black and white pixels, respectively, for the screen. You can modify its contents freely because it is not used in any Xlib function.

DefaultRootWindow(*display*)

Window XDefaultRootWindow(*display*)
    Display **display*;

Both return the root window for the default screen.

DefaultScreen(*display*)

int XDefaultScreen(*display*)
    Display **display*;

Both return the default screen number referenced in the XOpenDisplay routine. This macro and function should be used to retrieve the screen number in applications that will use only a single screen.

DefaultVisual(*display*, *screen_number*)

Visual *XDefaultVisual(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the default visual type for the specified screen. See Section 3.1 for further information about visual types.

DisplayCells(*display*, *screen_number*)

int XDisplayCells(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the number of entries in the default colormap.

DisplayPlanes(*display*, *screen_number*)

int XDisplayPlanes(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the depth of the root window of the specified screen. See the glossary for a discussion of depth.

DisplayString(*display*)

char *XDisplayString(*display*)
    Display **display*;

Both return the string that was passed to XOpenDisplay when the current display was opened. On UNIX-based systems, if the passed string was NULL, this macro returns the value of the DISPLAY environment variable when the current display was opened. This macro is useful to applications that invoke the fork system call and want to open a new connection to the same display from the child process.

ImageByteOrder(*display*)

int XImageByteOrder(*display*)
    Display **display*;

Both specifies the required byte order for images for each scanline unit in XYFormat (bitmap) or for each pixel value in ZFormat. The macro and function can return either LSBFirst or MSBFirst.

LastKnownRequestProcessed(*display*)

int XLastKnownRequestProcessed(*display*)
    Display **display*;

Both extract the full serial number of the last request known by Xlib to have been processed by the X server. This number is automatically set by Xlib when replies, events, and errors are received.

NextRequest(*display*)

int XNextRequest(*display*)
    Display **display*;

Both extract the full serial number that is to be used for the next request. Serial numbers are maintained separately for each display connection.

ProtocolVersion(*display*)

int XProtocolVersion(*display*)
    Display **display*;

Both return the major version number (11) of the X protocol associated with the connected display.

ProtocolRevision(*display*)

int XProtocolRevision(*display*)
    Display **display*;

Both return the minor protocol revision number of the X server.

QLength(*display*)

int XQLength(*display*)
    Display **display*;

Both return the length of the event queue for the connected display. Note that there may be more events that have not been read into the queue yet.

RootWindow(*display*, *screen_number*)

Window XRootWindow(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the root window. This macro and function are useful with functions that take a parent window as an argument.

ScreenCount(*display*)

int XScreenCount(*display*)
    Display **display*;

Both return the number of available screens.

ServerVendor(*display*)

char *XServerVendor(*display*)
    Display **display*;

Both return a pointer to a null-terminated string that provides some identification of the owner of the X server implementation.

VendorRelease(*display*)

int XVendorRelease(*display*)
    Display **display*;

Both return a number related to a vendor's release of the X server.

2.2.2. Image Format Macros

Applications are required to present data to the display server in a format that the server demands. To help simplify applications, most of the work required to convert the data is provided by Xlib. See Sections 6.7 and 10.9 for further information about these utility functions.

The following lists the C language macros, their corresponding function equivalents that are for other language bindings, and what data they both return for the specified server and screen. These are often used by toolkits as well as simple applications.

BitmapBitOrder(*display*)

int XBitmapBitOrder(*display*)
    Display **display*;

Within each bitmap unit, the leftmost bit in the bitmap as displayed on the screen is either the least or most significant bit in the unit. This macro and function can return either LSBFirst or MSBFirst.

BitmapPad(*display*)

int XBitmapPad(*display*)
    Display **display*;

Each scanline must be padded to a multiple of bits returned by this macro or function.

BitmapUnit(*display*)

int XBitmapUnit(*display*)
    Display **display*;

Both return the size of a bitmap's scanline unit in bits. The scanline is calculated in multiples of this value. It is always less than the bitmap scanline pad.

DisplayHeightMM(*display*, *screen_number*)

int XDisplayHeightMM(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the height of the specified screen in millimeters.

DisplayWidth(*display*, *screen_number*)

int XDisplayWidth(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return an integer that describes the width of the screen in pixels.

DisplayWidthMM(*display*, *screen_number*)

int XDisplayWidthMM(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return the width of the specified screen in millimeters.

2.2.3. Screen Information Macros

The following lists the C language macros, their corresponding function equivalents that are for other language bindings, and what data they both can return. These macros or functions all take a pointer to the appropriate screen structure.

BlackPixelOfScreen(*screen*)

unsigned long XBlackPixelOfScreen(*screen*)
    Screen **screen*;

Both return the black pixel value of the specified screen.

WhitePixelOfScreen(*screen*)

unsigned long XWhitePixelOfScreen(*screen*)
    Screen **screen*;

Both return the white pixel value of the specified screen.

CellsOfScreen(*screen*)

```
int XCellsOfScreen(screen)
    Screen *screen;
```

Both return the number of colormap cells of the specified screen.

DefaultColormapOfScreen(*screen*)

```
Colormap XDefaultColormapOfScreen(screen)
    Screen *screen;
```

Both return the default colormap of the specified screen.

DefaultDepthOfScreen(*screen*)

```
int XDefaultDepthOfScreen(screen)
    Screen *screen;
```

Both return the default depth of the specified screen.

DefaultGCOfScreen(*screen*)

```
GC XDefaultGCOfScreen(screen)
    Screen *screen;
```

Both return the default graphics context (GC) of the specified screen.

DefaultScreenOfDisplay(*display*)

```
Screen *XDefaultScreenOfDisplay(display)
    Display *display;
```

Both return the default screen of the specified display.

DefaultVisualOfScreen(*screen*)

```
Visual *XDefaultVisualOfScreen(screen)
    Screen *screen;
```

Both return the default visual of the specified screen. See Section 3.1 for information on visual types.

DoesBackingStore(*screen*)

```
int XDoesBackingStore(screen)
    Screen *screen;
```

Both return a value indicating whether the screen supports backing stores. The value returned can be one of WhenMapped, NotUseful, or Always. See Section 3.2.4 for a discussion of the backing store.

DoesSaveUnders(*screen*)

```
Bool XDoesSaveUnders(screen)
    Screen *screen;
```

Both return a Boolean value indicating whether the screen supports save unders. If True, the screen supports save unders. If False, the screen does not support save unders. See Section 3.2.6 for a discussion of the save under.

DisplayOfScreen(*screen*)

```
Display *XDisplayOfScreen(screen)
    Screen *screen;
```

Both return the display of the specified screen.

EventMaskOfScreen(*screen*)

```
long XEventMaskOfScreen(screen)
    Screen *screen;
```

Both return the initial root event mask for the specified screen.

HeightOfScreen(*screen*)

```
int XHeightOfScreen(screen)
    Screen *screen;
```

Both return the height of the specified screen.

HeightMMOfScreen(*screen*)

```
int XHeightMMOfScreen(screen)
    Screen *screen;
```

Both return the height of the specified screen in millimeters.

MaxCmapsOfScreen(*screen*)

```
int XMaxCmapsOfScreen(screen)
    Screen *screen;
```

Both return the maximum number of colormaps supported by the specified screen.

MinCmapsOfScreen(*screen*)

```
int XMinCmapsOfScreen(screen)
    Screen *screen;
```

Both return the minimum number of colormaps supported by the specified screen.

PlanesOfScreen(*screen*)

int XPlanesOfScreen(*screen*)
    Screen **screen*;

Both return the number of planes in the specified screen.

RootWindowOfScreen(*screen*)

Window XRootWindowOfScreen(*screen*)
    Screen **screen*;

Both return the root window of the specified screen.

ScreenOfDisplay(*display*, *screen_number*)

Screen *XScreenOfDisplay(*display*, *screen_number*)
    Display **display*;
    int *screen_number*;

Both return a pointer to the screen of the specified display.

WidthOfScreen(*screen*)

int XWidthOfScreen(*screen*)
    Screen **screen*;

Both return the width of the specified screen.

WidthMMOfScreen(*screen*)

int XWidthMMOfScreen(*screen*)
    Screen **screen*;

Both return the width of the specified screen in millimeters.

2.3. Generating a NoOperation Protocol Request

To execute a NoOperation protocol request, use XNoOp.

XNoOp(*display*)
    Display **display*;

display Specifies the connection to the X server.

The XNoOp function sends a NoOperation protocol request to the X server, thereby exercising the connection. It does not flush the output buffer.

2.4. Freeing Client-Created Data

To free any in-memory data that was created by an Xlib function, use XFree.

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

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

The XFree function is a general purpose Xlib routine that frees the specified data.

2.5. Closing the Display

To close or disconnect a display from the X server, use XCloseDisplay.

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

display Specifies the connection to the X server.

The XCloseDisplay function closes the connection to the X server for the display specified in the Display structure. The XCloseDisplay function 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 closedown mode of the resource has been changed (see XSetCloseDownMode). Therefore, these windows, resource IDs, and other resources should never be referenced again. In addition, this function discards any output requests that have been buffered but have not yet been sent. Because these operations automatically (implicitly) occur if a process exits, you normally do not have to call XCloseDisplay explicitly.

2.6. X Server Connection Close Operations

When the X server's connection to a client is closed, either by an explicit call to XCloseDisplay or by a process that exits, the X server performs these automatic operations:

- All selections (see XSetSelectionOwner) owned by the client are disowned.
- Performs an XUngrabPointer and XUngrabKeyboard if the client application has actively grabbed the pointer or the keyboard. These functions are described in Chapter 7.
- Performs an XUngrabServer if the client has grabbed the server. This function is described in Chapter 7.
- Releases all passive grabs made by the client application.
- Marks all resources (including colormap entries) allocated by the client application either as permanent or temporary, depending on whether the close_mode argument is either RetainPermanent or RetainTemporary. However, this does not prevent other client applications from explicitly destroying the resources. (See below and XSetCloseDownMode in Chapter 7 for further information.)

When the mode is DestroyAll, the X server destroys all of a client application's resources as follows:

- Examines each window in the client's save-set to determine if it is an inferior (subwindow) of a window created by the client. (The save-set is a list of other clients windows, and windows in this list are referred to as a save-set window.) If so, the X server reparents the save-set window to the closest ancestor such that the save-set window is not an inferior of a window created by the client.
- Performs a MapWindow request on the save-set window if the save-set window is unmapped. The X server does this even if the save-set window was not an inferior of a window created by the client.
- Examines each window in the client's save-set, and destroys all windows created by the client.
- Performs the appropriate free request on each nonwindow resource created by the client in the server (for example, Font, Pixmap, Cursor, Colormap, and GContext).
- Frees all colors and colormap entries allocated by a client application.

Additional processing occurs when the last connection to the X server closes. An X server goes through a cycle of having no connections and having some connections. When the last connection to the X server closes as a result of a connection closing with the close_mode of DestroyAll (that is, the X server reverts to the state of having no connections), the X server:

- Resets its state, as if it had just been started. The X server begins by destroying all lingering resources from clients that have terminated in RetainPermanent or RetainTemporary mode.
- Deletes all but the predefined atom identifiers.
- Deletes all properties on all root windows. See Chapter 4 for information about properties.
- Resets all device maps and attributes (for example, key click, bell volume, and acceleration) and the access control list.
- Restores the standard root tiles and cursors.
- Restores the default font path.
- Restores the input focus to state PointerRoot.

However, the X server does not reset if you close a connection with a close_down mode argument set to RetainPermanent or RetainTemporary.

Chapter 3

Window Functions

In the X Window System, a window is a rectangular area on the screen that lets you view graphical output. Client applications can display overlapping and nested windows on one or more screens that are driven by X servers on one or more machines. Clients who want to create windows must first connect their program to the X server by calling the Xlib function XOpenDisplay. This Chapter begins with a discussion of visual types and window attributes. The Chapter continues with a discussion of the Xlib functions you can use to:

- Create windows
- Destroy windows
- Map windows
- Unmap windows
- Configure windows
- Change the stacking order
- Change window attributes
- Translate window coordinates

Note that it is vital that your application conform to the established conventions for communicating with window managers for it to work well with the various window managers in use. See the discussion in Section 9.1 for more information. Toolkits generally adhere to these conventions for you, relieving you of the burden. Toolkits also often supersede many routines in this Chapter with versions of their own. Refer to the documentation for the toolkit you are using for more information.

This Chapter also identifies the window actions that may generate events. See Chapter 8 for a complete discussion of events.

3.1. Visual Types

On some high-end displays, it may be possible to deal with color resources in more than one way. For example, you may be able to deal with the display either as a 12-bit display with arbitrary mapping of pixel to color (pseudo-color) or as a 24-bit display with 8 bits of the pixel dedicated for red, green, and blue. These different ways of dealing with the visual aspects of the display are called Visuals. For each screen of the display, there may be a list of valid visual types supported at different depths of the display. Because there are default windows and visual types defined for each screen, most simple applications need not deal with this complexity. Xlib provides macros and functions that return the default root window, the default depth of the default root window, and the default visual type. See Chapter 2 for information on these macros and functions. See XMatchVisualInfo in Chapter 10 for information about how to find the visual type you need.

Xlib uses a Visual structure that contains information about the possible color mapping. The members of this structure pertinent to this discussion are class, red_mask, green_mask, blue_mask, bits_per_rgb, and map_entries. The class member specifies the possible visual classes of the screen. It can be one of the constants StaticGray, StaticColor, TrueColor, GrayScale, PseudoColor, or DirectColor.

Conceptually, as each pixel is read out of memory, it goes through a lookup stage by indexing into a colormap. Colormaps can be manipulated arbitrarily on some hardware, in limited ways on other hardware, and not at all on yet other hardware. The visual types affect the colormap and the RGB values in the following ways:

- For PseudoColor, a pixel value indexes a colormap to produce independent RGB values, and the RGB values can be changed dynamically.
- GrayScale is treated the same as PseudoColor, except the primary which drives the screen is undefined. Thus, the client should always store the same value for red, green, and blue in the colormaps.
- For DirectColor, a pixel value is decomposed into separate RGB subfields, and each subfield separately indexes the colormap for the corresponding value. The RGB values can be changed dynamically.
- TrueColor is treated the same as DirectColor, except the colormap has predefined read-only RGB values. These RGB values are server-dependent, but provide (near-)linear ramps in each primary.
- StaticColor is treated the same as PseudoColor, except the colormap has predefined read-only server-dependent RGB values.
- StaticGray is treated the same as StaticColor, except the red, green, and blue values are equal for any single pixel value, thus resulting in shades of gray. StaticGray with a two-entry colormap can be thought of as monochrome.

The red_mask, green_mask, and blue_mask members are only defined for DirectColor and TrueColor. Each has one contiguous set of bits with no interSections. The bits_per_rgb member specifies the log base 2 of the approximate number of distinct color values (individually) of red, green, and blue. Actual RGB values are unsigned 16 bit numbers. The map_entries member defines the number of available colormap entries in a newly created colormap. For DirectColor and TrueColor, this will be the size of an individual pixel subfield. The following concepts may serve to make the explanation of Visual types clearer. The screen can be color or grayscale. The screen can have a colormap that is writable or read-only. A screen can also have a colormap whose indices are decomposed into separate RGB pieces, provided one is not on a grayscale screen. This leads to the following diagram:

```
                     Color       GrayScale
                  R/O    R/W    R/O   R/W
                  +-------------------------------+
Undecomposed      | Static | Pseudo | Static | Gray  |
 Colormap         | Color  |Color   | Gray   | Scale |
                  +-------------------------------+
 Decomposed       | True   | Direct |
 Colormap         | Color  | Color  |
                  +---------------+
```

3.2. Window Attributes

All windows have a border width of zero or more pixels, an optional background, an input mask, an event suppression mask, and a property list. The window border and background can be a solid color or a pattern, called a tile. All windows except the root have a parent and are clipped by their parent. If a window is stacked on top of another window, it obscures that other window for the purpose of input. If a window has a background (almost all do), it obscures the other window for purposes of output. Attempts to output to the obscured area will do nothing, and no input events (for example, pointer motion) will be generated for the obscured area.

InputOnly windows only have the following attributes:

- win_gravity
- event_mask
- do_not_propagate_mask
- override_redirect

- cursor

InputOnly windows are used for controlling input events in situations where full-fledged windows are unnecessary. A BadMatch error is generated if you specifies any other attributes for an InputOnly window.

Windows have borders of a programmable width and pattern as well as a background pattern or tile. Pixels can be used for solid colors. In a program, you refer to the window using its resource ID of type Window. The background and border pixmaps may be destroyed immediately after creating the window if no further explicit references to them are to be made.

A window's background pattern can be either a solid color or a pattern. The pattern can either be relative to the parent or absolute. If relative to the parent, the pattern will be shifted appropriately to match the parent window. If absolute, the pattern will be positioned in the window independently of the parent window.

When windows are first created, they are not visible on the screen. Any output to a window not visible (not mapped) on the screen will be discarded. An application may wish to create a window long before it is mapped to the screen. When a window is eventually mapped to the screen (using XMapWindow), the X server will generate an exposure event for the window.

A window manager may override your choice as to size, border width, and position for a window. Your program must be prepared to use the actual size and position of the top window, which is reported when the window is first mapped. It is not acceptable for a client application to resize itself unless in direct response to a human command to do so. Instead, your program should either use the space given to it, or, if the space is too small for any useful work, your program might ask the user to resize the window. The border of your top-level windows are considered fair game for window managers.

The following symbols and the XSetWindowAttributes structure are used in the functions that follow.

```
#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)


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

The XSetWindowAttributes structure members are discussed in the following Sections.

3.2.1. The background_pixmap and background_pixel Members

The background_pixmap member specifies the pixmap to be used for a window's background. This pixmap can be of any size, although some sizes may be faster than others. The background_pixel member specifies a pixel value used to paint a window's background in a single color.

You can set the background_pixmap member to a pixmap, None, or ParentRelative. The default value is None. You can set the background_pixel to any pixel value. The default value is undefined. If you specifies a background_pixel, it overrides either the default background_pixmap or any value you may have set in the background_pixmap member. All pixels in the background of the window will be set to this value.

If you set the background_pixmap, it overrides the default background_pixmap. The background_pixmap and the window must have the same depth. Otherwise, a BadMatch error is returned. If you set background_pixmap to None, the window has no defined background. If the parent window has a background_pixmap of None, the window will also have a background_pixmap of None. If you set the background_pixmap to ParentRelative:

- The parent window's background_pixmap is used, but the child window must have the same depth as its parent. Otherwise, a BadMatch error is returned.
- A copy of the parent window's background_pixmap is not made. The parent's background_pixmap is examined each time the child window background_pixmap is required.
- The background tile origin always aligns with the parent window's background tile origin. Otherwise, the background tile origin is always the child window origin.

Setting a new background, whether by setting background_pixmap or background_pixel, overrides any previous border. The background_pixmap can be freed immediately if no further explicit reference is made to it (the X server will keep a copy to use when needed). If you later draw into the pixmap used for the background, X does not predict what happens because the X implementation is free to either make a copy of the pixmap or just use the same pixmap.

When no valid contents are available for regions of a window, and either the regions are visible, or the server is maintaining backing store, the server automatically tiles the regions with the window's background, unless the window has a background of None. If the background is None, the previous screen contents are simply left in place, if the contents come from an inferior window of the same depth. Otherwise, the initial contents of the exposed regions are undefinded. Exposure events are then generated for the regions, even if the background_pixmap is None. See Chapter 8 for a discussion of exposure event processing.

3.2.2. The border_pixmap and border_pixel Members

The border_pixmap member specifies the pixmap to be used for a window's border. This pixmap can be of any size, although some sizes may be faster than others. The border_pixel member specifies a pixmap of undefined size be used for a window's border. The border tile origin is always the same as the background tile origin.

You can also set border_pixmap to CopyFromParent. In this case, the pixmap used for the border will be a copy of the parent window's border pixmap. The default value is

CopyFromParent. You can set the border_pixel to any pixel value. The default value is undefined.

If you set a border_pixmap value, it overrides the default border_pixmap. The border_pixmap and the window must have the same depth. Otherwise, a BadMatch error is returned. If you set the border_pixmap to CopyFromParent, the parent window's border_pixmap is copied. Subsequent changes to the parent window's border attribute do not affect the child window. However, the child window must have the same depth as the parent window. Otherwise, a BadMatch error is returned.

The border_pixmap can be freed immediately if no further explicit reference is made to it. If you later draw into the pixmap used for the border, X does not predict what happens because the X implementation is free to either make a copy of the pixmap or use the same pixmap each time the window border is repainted. If you specifies a border_pixel, it overrides either the default border_pixmap or any value you may have set in the border_pixmap member. All pixels in the window's border will be set to the border_pixel value. Setting a new border, whether by setting border_pixel or by setting border_pixmap overrides any previous border.

Output to a window is always clipped to the inside of the window. Therefore, graphics operations never affect the window border. Borders are added to the window size specified.

3.2.3. The bit_gravity and win_gravity Members

Bit gravity defines which region of the window should be retained when the window is resized. The default value for the bit_gravity member is the constant ForgetGravity. Window gravity allows you to define how the window should be repositioned if its parent is resized. The default value for the win_gravity member is the constant NorthWestGravity.

If the inside width or height of a window is not changed, and if the window is moved or its border is changed, then the contents of the window are not lost but move with the window. Changing the inside width or height of the window causes its contents to be moved or lost, depending on the bit_gravity of the window, and causes children to be reconfigured, depending on their win_gravity. For a change of width and height, the (x, y) pairs are defined:

Gravity Direction	Coordinates
NorthWestGravity	(0, 0)
NorthGravity	(Width/2, 0)
NorthEastGravity	(Width, 0)
WestGravity	(0, Height/2)
CenterGravity	(Width/2, Height/2)
EastGravity	(Width, Height/2)
SouthWestGravity	(0, Height)
SouthGravity	(Width/2, Height)
SouthEastGravity	(Width, Height)

When a window with one of these bit_gravities is resized, the corresponding pair defines the change in position of each pixel in the window. When a window with one of these win_gravities has its parent window resized, the corresponding pair defines the change in position of the window within the parent. When a window is so repositioned, a GravityNotify event is generated.

A bit_gravity of StaticGravity indicates that the contents or origin should not move relative to the origin of the root window. If the change in size of the window is coupled with a change in position (x, y), then for bit_gravity the change in position of each pixel is (-x, -y), and for win_gravity the change in position of a child when its parent is so resized is (-x, -y). Note that StaticGravity still only takes effect when the width or height of the window is changed, not when the window is moved.

A bit_gravity of ForgetGravity indicates that the window's contents are always discarded after a size change, even if a backing_store or save_under has been requested. The window is tiled with its background and one or more exposure events are generated. If no background is defined, the existing screen contents are not altered. Some X servers may also ignore the specified bit_gravity and always generate exposure events.

A win_gravity of UnmapGravity is like NorthWest (the window is not moved), but the child is also unmapped when the parent is resized, and an UnmapNotify event is generated. A win_gravity of AntiGravity indicates all pixels should move radically outward from the center of the window.±

3.2.4. The backing_store Member

Some implementations of the X server may choose to maintain the contents of windows. If the X server maintains the contents of a window, the off-screen saved pixels are known as backing store. The backing_store member advises the X server on what to do with the contents of a window. You can set this member to NotUseful, WhenMapped, or Always. The default value is NotUseful.

A backing_store of WhenMapped advises the X server that maintaining contents of obscured regions when the window is mapped would be beneficial. In this case, the server may generate an exposure event when the window is created. A backing_store of Always advises the X server that maintaining contents even when the window is unmapped would be beneficial. Even if the window is larger than its parent, this is a request to the X server to maintain complete contents, not just the region within the parent window boundaries. While the X server maintains contents, exposure events normally will not be generated, but the X server may stop maintaining contents at any time. A backing_store of NotUseful advises the X server that maintaining contents is unnecessary, although some X implementations may still choose to maintain contents and, therefore, not generate exposure events.

When the contents of obscured regions of a window are being maintained, regions obscured by non-inferior windows are included in the destination (and source, when the window is the source) of graphics requests. However, regions obscured by inferior windows are not included.

3.2.5. The save_under Member

Some server implementations may preserve bits of windows under other windows. This is not the same as preserving the contents of a window for you. You may get better visual appeal if transient windows (for example, pop-up menus) request that the system preserve the bits under them, so the temporarily obscured applications do not have to repaint.

The default value for the save_under member is False. If save_under is True, the X server is advised that, when this window is mapped, saving the contents of windows it obscures would be beneficial.

3.2.6. The backing_planes and backing_pixel Members

The backing_planes member indicates (with one bits) which bit planes of the window hold dynamic data that must be preserved in backing store and during save unders. The default value for the backing_planes member is all ones. The backing_pixel specifies what values to use in planes not covered by backing_planes. The default value for the backing_pixel member is zero. The X server is free to save only the specified bit planes in the backing store or the save under and is free to regenerate the remaining planes with the specified pixel value. Any extraneous bits (that is, those beyond the specified depth of the window) in these values may be simply ignored. If you request backing store or save unders you should use these members to minimize the amount of off-screen memory required to store your window.

± (If you believe this statement, there is a bridge for sale.)

3.2.7. The event_mask and do_not_propagate_mask Members

The event_mask defines which events the client is interested in for this window (or, for some event types, inferiors of the window). The do_not_propagate_mask defines which events should not be propagated to ancestor windows when no client has the event type selected in this window. These masks are the bitwise inclusive OR of one or more of the valid event mask bits. You can specifies that no maskable events are reported by passing NoEventMask. The default value for these members is the empty set. See Section 8.3 for information on the event mask and events.

3.2.8. The override_redirect Member

To control window placement or to add decoration, a window manager often needs to intercept (''redirect'') any map or configure request. Pop-up windows, however, need to be mapped without a window manager getting in the way for quick response. You can control if a window is to ignore these structure control facilities by use of the override_redirect mask.

The default value for the override_redirect member is False. Override_redirect specifies whether map and configure requests on this window should override a SubstructureRedirectMask on the parent. Window managers use this information to avoid tampering with pop-up windows.

3.2.9. The colormap Member

The colormap member specifies which colormap, if any, best reflects the true colors of the window. The colormap must have the same visual type as the window. Otherwise, a BadMatch error is returned. X servers capable of supporting multiple hardware colormaps may use this information, and window managers may use it for XInstallColormap requests. If you set the colormap member to CopyFromParent, the parent window's colormap is copied and used by its child. The default value for the colormap member is CopyFromParent. Subsequent changes to the parent window's colormap attribute do not affect the child window. However, the child window must have the same visual type as the parent. Otherwise, a BadMatch error is returned. The parent window must not have a colormap of None. Otherwise, a BadMatch error is returned.

The colormap is copied by sharing the colormap object between the child and parent, not by making a complete copy of the colormap contents.

3.2.10. The cursor Member

If a cursor is specified, it will be used whenever the pointer is in the window. If None is specified, the parent's cursor will be used when the pointer is in the window, and any change in the parent's cursor will cause an immediate change in the displayed cursor. The default value for the cursor member is None. The cursor may be freed immediately if no further explicit reference to it is made by calling XFreeCursor. See Section 6.8.2 for further information.

3.2.11. Default Values for XSetWindowAttributes Members

The following table lists the default values for each member in the XSetWindowAttributes structure.

Member	Default Value
background_pixmap	None
background_pixel	Undefined
border_pixmap	CopyFromParent
border_pixel	Undefined
bit_gravity	ForgetGravity
win_gravity	NorthWestGravity
backing_store	NotUseful
backing_planes	All ones

Member	Default Value
backing_pixel	0 (zero)
save_under	False
event_mask	empty set
do_not_propagate_mask	empty set
override_redirect	False
colormap	CopyFromParent
cursor	None

3.3. Creating Windows

Xlib provides basic ways of creating windows. See the X Toolkit documentation for more information. If you create your own top level windows (direct children of the root window) the rules enumerated below must be observed for applications to interact reasonably across differing styles of window management.

You should never fight with a window manager for size or placement of your top-level window(s). Toolkits often supply routines specifically for creating and placing top level windows. If you do not use a toolkit, you should provide some standard information or ''hints'' to the window manager by using the utility functions described in Chapter 10.

The policy guidelines for window creation are:

- An application, by listening to the first exposure event, must be able to deal with whatever size window it gets, even if this means that the application just prints a message, like ''Please make me bigger,'' in its window.
- An application should only attempt to resize or move its top-level window in direct response to a user request. An application is free to resize or move the children of its top-level window as necessary. (Toolkits often have facilities for automatic relayout.) If a request to change the size of its top-level window fails, the application must not fight with the window manager.
- If an application does not use a toolkit that automatically sets standard window properties, that application should set these properties for the top-level window before mapping it. To set standard window properties for a top-level window, use XSetStandardProperties. See Chapter 9 for further information.

The low-level functions provided by Xlib to create an unmapped subwindow for a specified parent window are XCreateWindow and XCreateSimpleWindow. XCreateWindow is a more general function that allows you to set specific window attributes when you create it. XCreateSimpleWindow creates a window that inherits its attributes from its parent window. That is, you do not set specific attributes when you create a simple window.

The X server acts as if InputOnly windows do not exist for the purposes of graphics requests, exposure processing, and VisibilityNotify events. An InputOnly window cannot be used as a drawable (that is, as a source or destination for graphics requests). InputOnly and InputOutput windows act identically in other respects (properties, grabs, input control, and so on). Extension packages may define other classes of windows.

The definition for XCreateWindow is:

```
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;
```

display	Specifies the connection to the X server.
parent	Specifies the parent window ID.
x *y*	Specify the x and y coordinates. These coordinates are the top left outside corner of the created window's borders and are relative to the inside of the parent window's borders.
width *height*	Specify the width and height. These are the created window's inside dimensions. These dimensions do not include the created window's borders, which are entirely outside of the window. The dimensions must be nonzero. Otherwise, XCreateWindow generates a BadValue error.
border_width	Specifies, in pixels, the width of the created window's border. The border_width for an InputOnly window must be zero. Otherwise, a BadMatch error is returned.
depth	A depth of zero for class InputOutput or CopyFromParent means the depth is taken from the parent.
class	Specifies the created window's class. You can pass one of these constants: InputOutput, InputOnly, or CopyFromParent. A class of CopyFromParent means the class is taken from the parent.
visual	Specifies the visual type. A visual of CopyFromParent means the visual type is taken from the parent.
valuemask	Specifies which window attributes are defined in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero, the attributes are ignored and are not referenced.
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. See Section 3.2 for further information.

The XCreateWindow function creates an unmapped subwindow for a specified parent window, returns the window ID of the created window, and causes the X server to generate a CreateNotify event. The created window is placed on top in the stacking order with respect to siblings.

For class InputOutput, the visual type and depth must be a combination supported for the screen. Otherwise, XCreateWindow generates a BadMatch error. The depth need not be the same as the parent, but the parent must not be a window of class InputOnly. Otherwise, it generates a BadMatch error. For an InputOnly window the depth must be zero, and the visual must be one supported by the screen. If either of these conditions are not met, a BadMatch error is generated. The parent window, however, may have any depth and class. The only window attributes defined for InputOnly windows are win_gravity, event_mask, do_not_propagate_mask, override_redirect, and cursor. If you specifies any other window attribute for an InputOnly window, a BadMatch error is returned.

XCreateWindow can generate BadAlloc BadColor, BadCursor, BadMatch, BadPixmap, BadValue, and BadWindow errors.

To create an unmapped InputOutput subwindow of the specified parent window, use XCreateSimpleWindow.

```
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;
```

display	Specifies the connection to the X server.
parent	Specifies the parent window ID.
x *y*	Specify the x and y coordinates. These coordinates are the top left outside corner of the new window's borders and are relative to the inside of the parent window's borders.
width *height*	Specify the width and height. These are the created window's inside dimensions. These dimensions do not include the created window's borders, which are entirely outside of the window. The dimensions must be nonzero. Otherwise, XCreateSimpleWindow generates a BadValue error.
border_width	Specifies, in pixels, the width of the created window's border. The border_width for an InputOnly window must be zero. Otherwise, a BadMatch error is returned.
border	Specifies the border pixel value of the window.
background	Specifies the background pixel value of the window.

The XCreateSimpleWindow function creates an unmapped InputOutput subwindow for a specified parent window, returns the window ID of the created window, and causes the X server to generate a CreateNotify event. The created window is placed on top in the stacking order with respect to siblings. Any part of the window that extends outside its parent window will be clipped. XCreateSimpleWindow inherits its depth, class, and visual from its parent. All other window attributes have their default values.

The created window is not yet displayed (mapped) on the user's display. To display the window, call XMapWindow. The new window will initially use the same cursor as its parent. A new cursor may, of course, be defined for the new window by calling XDefineCursor. The window will not be visible on the screen unless it and all of its ancestors are mapped, and it is not obscured by any of its ancestors. The window is placed on top of the stacking order with respect to its siblings. In addition, the new window's cursor will be None.

XCreateSimpleWindow can generate BadAlloc, BadMatch, BadValue, and BadWindow errors.

3.4. Destroying Windows

Xlib provides functions with which you can destroy a window or destroy all subwindows of a window.

To destroy a window and all of its subwindows, use XDestroyWindow.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XDestroyWindow function destroys the specified window as well as all of its subwindows and causes the X server to generate a DestroyNotify event for each window. The window should never again be referenced. If the window specified by the w argument is mapped, it is unmapped automatically. The window and all of its inferiors are then destroyed, and a Destroy-Notify event is generated for each window. The ordering of the DestroyNotify events is such that for any given window being destroyed, DestroyNotify is generated on any inferiors of the window before being generated on the window itself. The ordering among siblings and across subhierarchies is not otherwise constrained. If the window you specified is a root window, no windows are destroyed. Destroying a mapped window will generate exposure events on other windows that were obscured by the window being destroyed.

XDestroyWindow can generate a BadWindow error.

To destroy all subwindows of a specified window, use XDestroySubwindows.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XDestroySubwindows function destroys all inferior windows of the specified window, in bottom to top stacking order. It causes the X server to generate a DestroyNotify event for each window. If any mapped subwindows were actually destroyed, XDestroySubwindows causes the X sever to generate exposure events on the specified window. This is much more efficient than deleting many windows one at a time because much of the work need only be performed once for all of the windows rather than for each window. The subwindows should never again be referenced. Note that by default, windows are destroyed when a connection is closed. See Section 2.6 for more information.

XDestroySubwindows can generate a BadWindow error.

3.5. Mapping Windows

A window manager may want to control the placement of subwindows. If Substruc-tureRedirectMask has been selected by a window manager on a parent window (usually a root window), a map request initiated by other clients on a child window is not performed, and the window manager would be sent a MapRequest event. However, if the override_redirect flag on the child had been set to True (usually only on pop-up menus), the map request would be per-formed.

A tiling window manager might decide to reposition and resize other client's windows and then decide to map the window at its final location. A window manager that wants to provide decora-tion might reparent the child into a frame first. See Section 3.2.7 and Chapter 8 for further infor-mation. Only a single client at a time can select for SubstructureRedirectMask.

Similarly, a single client can select for ResizeRedirectMask on a parent window. Then, any attempt to resize the window is suppressed, and the client (usually, a window manager) receives a ResizeRequest event. These mechanisms allow arbitrary placement policy to be enforced by an external window manager.

A window is considered mapped if a XMapWindow call has been made on it. It may not be visible on the screen for one of the following reasons:

- It is obscured by another opaque sibling window.
- One of its ancestors is not mapped.
- It is entirely clipped by an ancestor.

Exposure events will be generated for the window when part or all of it becomes visible on the screen. A client will only receive the exposure events if it has asked for them using XSelectInput. Windows retain their position in the stacking order when unmapped.

To map the specified window, use XMapWindow.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XMapWindow function maps the window and all of its subwindows which have had map requests. A subwindow will appear on the screen as long as all of its ancestors are mapped and not obscured by a sibling or are not clipped by an ancestor. Mapping a window that has an unmapped ancestor does not display the window but marks it as eligible for display when the ancestor becomes mapped. Such a window is called unviewable. When all its ancestors are mapped, the window becomes viewable and will be visible on the screen if it is not obscured by any sibling or ancestor. This function has no effect if the window is already mapped.

If the override_redirect member of the XSetWindowAttributes structure is False, and if some other client has selected SubstructureRedirectMask on the parent window, then the X server generates a MapRequest event, and the XMapWindow function does not map the window. Otherwise, the window is mapped, and the X server generates a MapNotify event.

If the window becomes viewable and no earlier contents for it are remembered, XMapWindow tiles the window with its background. If no background was defined for the window, the existing screen contents are not altered, and the X server generates one or more Expose events. If a backing_store was maintained while the window was unmapped, no Expose events are generated. If a backing_store will now be maintained, a full-window exposure is always generated. Otherwise, only visible regions may be reported. Similar tiling and exposure take place for any newly viewable inferiors.

If the window is an InputOutput window, XMapWindow generates Expose events on each InputOutput window that it causes to become displayed. If the client maps and paints the window, and if the client begins processing events, the window will be painted twice. To avoid this, the client should call XSelectInput for exposure events, and map the window. Then, the client should process input events normally. The event list will include Expose for each window that has appeared on the screen. The client's normal response to an Expose event should be to repaint the window. This method usually leads to simpler programs and to proper interaction with window managers.

XMapWindow can generate a BadWindow error.

To map and raise a window, use XMapRaised.

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

display	Specifies the connection to the X server.

w	Specifies the window ID.

The XMapRaised function essentially is similar to XMapWindow in that it maps the window and all of its subwindows which have had map requests. However, it also raises the specified window to the top of the stack. See the XMapWindow description for additional information.

XMapRaised can generate a BadWindow error.

To map all subwindows for a specified window, use XMapSubwindows.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XMapSubwindows function maps all subwindows for a specified window in top-to-bottom stacking order. The X server to generate an Expose event on each newly displayed window. This is much more efficient than mapping many windows one at a time, because the server needs only perform much of the work once for all of the windows rather than for each window.

XMapSubwindows can generate a BadWindow error.

3.6. Unmapping Windows

Xlib provides functions with which you can unmap a window or all subwindows.

To unmap a window, use XUnmapWindow.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XUnmapWindow function unmaps the specified window and causes the X server to generate an UnmapNotify event. If the specified window is already unmapped, XUnmapWindow has no effect. Normal exposure processing on formerly obscured windows is performed. Any child window will no longer be visible until another map call is made on the parent. In other words, the subwindows are still mapped but not visible until the parent is mapped. Unmapping a window will generate exposure events on windows that were formerly obscured by it and its children.

XUnmapWindow can generate a BadWindow error.

To unmap all subwindows for a specified window, use XUnmapSubwindows.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XUnmapSubwindows function unmaps all subwindows for the specified window in bottom to top stacking order. It causes the X server to generate an UnmapNotify event on each subwindow and exposure events on formerly obscured windows. Using this function is much more efficient than unmapping multiple windows one at a time, because the server needs only perform much of the work once for all of the windows rather than for each window.

XUnmapSubwindows can generate a BadWindow error.

3.7. Configuring Windows

Xlib provides functions with which you can move a window, resize a window, move and resize a window, or change a window's border width. The most general interface to configuring windows, XConfigureWindow, uses the XWindowChanges structure, which contains:

```
#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)

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

The x and y members specifies the x and y coordinates relative to the parent's origin and indicate the position of the upper-left, outer-corner of the window. The width and height members specifies the inside size of the window, not including the border. These arguments must be nonzero. Otherwise, a BadValue error is generated. Attempts to configure a root window have no effect.

The border_width member specifies the width of the border in pixels. Note that changing just the border_width leaves the outer-left corner of the window in a fixed position, but moves the absolute position of the window's origin. A BadMatch error is generated if you attempt to make the border_width of an InputOnly window nonzero.

The sibling member specifies the sibling window for stacking operations. The stack_mode member can be one of these constants: Above, Below, TopIf, BottomIf, or Opposite.

If the override_redirect attribute of the window is False, and if some other client has selected SubstructureRedirectMask on the parent, then the X server generates a ConfigureRequest event, and no further processing is performed. Otherwise, if some other client has selected ResizeRedirectMask on the window and the inside width or height of the window is being changed, a ResizeRequest event is generated, and the current inside width and height are used instead in the following. Note that the override_redirect attribute of the window has no effect on ResizeRedirectMask and that SubstructureRedirectMask on the parent has precedence over ResizeRedirectMask on the window.

When the geometry of the window is changed as specified, the window is restacked among siblings, and a ConfigureNotify event is generated if the state of the window actually changes. X generates GravityNotify events after generating ConfigureNotify events. If the inside width or height of the window has actually changed, then children of the window are affected as described below.

If a window's size actually changes, the window's subwindows may move according to their window gravity. Depending on the window's bit gravity, the contents of the window also may be moved. See Section 3.2.3 for further information.

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

The restack check (specifically, the computation for BottomIf, TopIf, and Opposite) is performed with respect to the window's final size and position (as controlled by the other arguments to the request), not its initial position. It is an error if a sibling is specified without a stack_mode.

If a sibling and a stack_mode are specified, the window is restacked as follows:

Above	The window is placed just above the sibling.
Below	The window is placed just below the sibling.
TopIf	If the sibling occludes the window, the window is placed at the top of the stack.
BottomIf	If the window occludes the sibling, the window is placed at the bottom of the stack.
Opposite	If the sibling occludes the window, the window is placed at the top of the stack. Otherwise, if the window occludes the sibling, the window is placed at the bottom of the stack.

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

Above	The window is placed at the top of the stack.
Below	The window is placed at the bottom of the stack.
TopIf	If any sibling occludes the window, the window is placed at the top of the stack.
BottomIf	If the window occludes any sibling, the window is placed at the bottom of the stack.
Opposite	If any sibling occludes the window, the window is placed at the top of the stack. Otherwise, if the window occludes any sibling, the window is placed at the bottom of the stack.

To configure a window's size, location, stacking, or border, use XConfigureWindow.

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

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window to be reconfigured.
value_mask	Specifies which values are to be set using information in the values structure. This mask is the bitwise inclusive OR of the valid change window values bits.
values	Specifies a pointer to the structure XWindowChanges.

The XConfigureWindow function uses the values specified in the XWindowChanges structure to reconfigure a window's size, position, border, and stacking order. The stacking order of the window is controlled by the function's arguments. Values not specified are taken from the existing geometry of the window.

A BadMatch error is generated if a sibling is specified without a stack_mode or if the window is not actually a sibling. Note that the computations for BottomIf, TopIf, and Opposite are performed with respect to the window's final geometry (as controlled by the other arguments passed to XConfigureWindow), not its initial geometry. Any backing-store contents of the window, its inferiors, and other newly visible windows are either discarded or changed to reflect the current screen contents (implementation dependent).

XConfigureWindow can generate BadMatch, BadValue, and BadWindow errors.

To move a window without changing its size, use XMoveWindow.

```
XMoveWindow(display, w, x, y)
    Display *display;
    Window w;
    int x, y;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window to be moved.
x *y*	Specify the x and y coordinates. These coordinates define the new location of the top left pixel of the window's border or the window itself, if it has no border.

The XMoveWindow function moves the specified window to the specified x and y coordinates. This function does not change the window's size, does not raise the window, and does not change the mapping state of the window. Moving a mapped window may or may not lose its contents depending on:

- If its background_pixmap attribute has the value ParentRelative.
- If the window is obscured by nonchildren, and no backing store exists.

If the contents of the window are lost, exposure events will be generated for the window and any mapped subwindows. Moving a mapped window will generate exposure events on any formerly obscured windows.

If the override_redirect attribute of the window is False and some other client has selected SubstructureRedirectMask on the parent, a ConfigureRequest event is generated, and no further processing is performed. Otherwise, the window is moved.

XMoveWindow can generate a BadWindow error.

To change a window's size without changing the upper-left coordinate, use XResizeWindow.

```
XResizeWindow(display, w, width, height)
    Display *display;
    Window w;
    unsigned int width, height;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
width *height*	Specify the width and height. These are the dimensions of the window after the call completes.

The XResizeWindow function changes the inside dimensions of the specified window, not including its borders. This function does not change the window's upper-left coordinate or the origin and does not raise the window. Changing the size of a mapped window may lose its contents and generate an Expose event. If a mapped window is made smaller, exposure events will be generated on windows that it formerly obscured.

If the override_redirect attribute of the window is False and some other client has selected SubstructureRedirectMask on the parent, a ConfigureRequest event is generated, and no further processing is performed.

XResizeWindow can generate a BadWindow error.

To change the size and location of a window, use XMoveResizeWindow.

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

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window to be reconfigured.
x *y*	Specify the x and y coordinates. These coordinates define the new position of the window relative to its parent.
width *height*	Specify the width and height. These arguments define the interior size of the window.

The XMoveResizeWindow function changes the size and location of the specified window without raising it. Moving and resizing a mapped window may generate an Expose event on the window. Depending on the new size and location parameters, moving and resizing a window may generate exposure events on windows that the window formerly obscured.

If the override_redirect attribute of the window is False and some other client has selected SubstructureRedirectMask on the parent, then a ConfigureRequest event is generated, and no further processing is performed. Otherwise, the window size and location is changed.

XMoveResizeWindow can generate BadMatch, BadValue, and BadWindow errors.

To change the border width of a window, use XSetWindowBorderWidth.

```
XSetWindowBorderWidth(display, w, width)
    Display *display;
    Window w;
    unsigned int width;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
width	Specifies the width of the window border.

The XSetWindowBorderWidth function sets the specified window's border width to the specified width.

XSetWindowBorderWidth can generate BadValue and BadWindow errors.

3.8. Changing Window Stacking Order

Xlib provides functions with which you can raise, lower, circulate, or restack windows.

To raise a window so that no sibling window obscures it, use XRaiseWindow.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XRaiseWindow function raises the specified window to the top of the stack so that no sibling window obscures it. If the windows are regarded as overlapping sheets of paper stacked on a desk, then raising a window is analogous to moving the sheet to the top of the stack but leaving its x and y location on the desk constant. Raising a mapped window may generate exposure events for the window and any mapped subwindows that were formerly obscured.

If the override_redirect attribute of the window is False and some other client has selected SubstructureRedirectMask on the parent, a ConfigureRequest event is generated, and no processing is performed. Otherwise, the window is raised.

XRaiseWindow can generate a BadWindow error.

To lower a window so that it does not obscure any sibling windows, use XLowerWindow.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XLowerWindow function lowers the specified window to the bottom of the stack so that it does not obscure any sibling windows. If the windows are regarded as overlapping sheets of paper stacked on a desk, then lowering a window is analogous to moving the sheet to the bottom of the stack but leaving its x and y location on the desk constant. Lowering a mapped window will generate exposure events on any windows it formerly obscured.

If the override_redirect attribute of the window is False and some other client has selected SubstructureRedirectMask on the parent, a ConfigureRequest event is generated, and no processing is performed. Otherwise, the window is lowered to the bottom of the stack.

XLowerWindow can generate a BadWindow error.

To circulate a subwindow up or down, use XCirculateSubwindows.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.
direction	Specifies the direction (up or down) that you want to circulate the window. You can pass one of these constants: RaiseLowest or LowerHighest.

The XCirculateSubwindows function circulates the specified subwindow in the specified direction: RaiseLowest or LowerHighest. If some other client has selected SubstructureRedirectMask on the window, a CirculateRequest event is generated, and no further processing is performed. Otherwise, the processing described in the following paragraph is performed, and if the window is actually restacked, the X server generates a CirculateNotify event.

If you specifies RaiseLowest, this function raises the lowest mapped child (if any) that is occluded by another child to the top of the stack. If you specifies 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.

XCirculateSubwindows can generate BadValue and BadWindow errors.

To raise the lowest mapped child of an occluded window, use XCirculateSubwindowsUp.

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

display	Specifies the connection to the X server.

w Specifies the window ID.

The XCirculateSubwindowsUp function raises the lowest mapped child of the specified window that is partially or completely occluded by another child. Completely unobscured children are not affected. This is a convenience routine equivalent to XCirculateWindow (display, window, RaiseLowest).

XCirculateSubwindowsUp can generate a BadWindow error.

To lower the highest mapped child of a window that partially or completely occludes another child, use XCirculateSubwindowsDown.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XCirculateSubwindowsDown function lowers the highest mapped child of the specified window that partially or completely occludes another child. Completely unobscured children are not affected. This is a convenience routine equivalent to XCirculateWindow (display, window, LowerHighest).

XCirculateSubwindowsDown can generate a BadWindow error.

To restack a set of windows from top to bottom, use XRestackWindows.

```
XRestackWindows(display, windows, nwindows);
    Display *display;
    Window windows[];
    int nwindows;
```

display	Specifies the connection to the X server.
windows	Specifies an array containing the windows to be restacked. All the windows must have the same parent.
nwindows	Specifies the number of windows to be restacked.

The XRestackWindows function restacks the windows in the order specified, from top to bottom. The stacking order of the first window in the windows array will be unaffected, but the other windows in the array will be stacked underneath the first window in the order of the array. The stacking order of the other windows is not affected.

If the override_redirect attribute of a window is False and some other client has selected SubstructureRedirectMask on the parent, ConfigureRequest events are generated for each window whose override_redirect is not set, and no further processing is performed. Otherwise, the windows will be restacked in top to bottom order.

XRestackWindows can generate a BadWindow error.

3.9. Changing Window Attributes

Xlib provides functions with which you can set window attributes. XChangeWindowAttributes is the more general function that allows you to set one or more window attributes provided by the XSetWindowAttributes structure. See Section 3.2 for descriptions of these window attributes. The other functions described in this Section allow you to set one specific window attribute, such as a window's background.

To change one or more window attributes, use XChangeWindowAttributes.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.
valuemask	Specifies which window attributes are defined in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero, the attributes are ignored and are not referenced. The values and restrictions are the same as for XCreateSimpleWindow and XCreateWindow.
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.

Depending on the valuemask, the XChangeWindowAttributes function uses the window attributes in the XSetWindowAttributes structure to change the specified window attributes. Changing the background does not cause the window contents to be changed. To repaint the window and its background, use XClearWindow. (See Section 6.2 for further information.) Setting the border or changing the background such that the border tile origin changes causes the border to be repainted. Changing the background of a root window to None or ParentRelative restores the default background pixmap. Changing the border of a root window to CopyFromParent restores the default border pixmap. Changing the win_gravity does not affect the current position of the window. Either changing the backing_store of an obscured window to WhenMapped or Always, or changing the backing_planes, backing_pixel, or save_under of a mapped window may have no immediate effect.

Multiple clients can select input on the same window. If this is the case, their event masks are maintained separately. When an event is generated, it will be reported to all interested clients. However, at most, one client at a time can select for SubstructureRedirectMask, ResizeRedirectMask, and ButtonPressMask. If a client attempts to select any of these event masks and some other client has already selected it, 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 of a window (that is, defining a new map, not changing the contents of the existing map) generates a ColormapNotify event. Changing the colormap of a visible window may have no immediate effect on the screen because the map may not be installed. See XInstallColormap in Chapter 7. Changing the cursor of a root window to None restores the default cursor. Whenever possible, you are encouraged to share colormaps.

XChangeWindowAttributes can generate BadAccess, BadColor, BadCursor, BadMatch, BadPixmap, BadValue, and BadWindow errors.

To set the background of a specified window to the specified pixel, use XSetWindowBackground.

XSetWindowBackground(*display*, *w*, *background_pixel*)
    Display **display*;
    Window *w*;
    unsigned long *background_pixel*;

display	Specifies the connection to the X server.
w	Specifies the window ID.
background_pixel	Specifies the pixel of the background. This pixel value determines which entry in the color map is used.

The XSetWindowBackground function sets the background pixel of the window to the pixel value you specify. Changing the background does not cause the window contents to be changed. XSetWindowBackground uses a pixmap of undefined size filled with the color associated with the pixel value you passed to the background_pixel argument. This can not be performed on an InputOnly window. An error will result if you try to change the background of an InputOnly window.

XSetWindowBackground can generate BadMatch and BadWindow errors.

To set the background of a specified window to the specified pixmap, use XSetWindowBackgroundPixmap.

```
XSetWindowBackgroundPixmap(display, w, background_pixmap)
    Display *display;
    Window w;
    Pixmap background_pixmap;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
background_pixmap	Specifies the background pixmap. If a Pixmap ID is specified, the background is painted with this pixmap. If None, no background is painted. If ParentRelative, the parent's pixmap is used.

The XSetWindowBackgroundPixmap function sets the background pixmap of the window to the pixmap you specify. If no background Pixmap is specified, the background pixmap of the window's parent is used. On the root window, the default background will be restored. The background Pixmap can immediately be freed if no further explicit references to it are to be made. This can not be performed on an InputOnly window. An error will result if you try to change the background of an InputOnly window.

XSetWindowBackgroundPixmap can generate BadColor, BadMatch, BadPixmap, and BadWindow errors.

Note

> XSetWindowBackground and XSetWindowBackgroundPixmap do not change the current contents of the window, and you may wish to clear and repaint the screen after these functions because they will not repaint the background you just set.

To change and repaint a window's border to the specified pixel, use XSetWindowBorder.

```
XSetWindowBorder(display, w, border_pixel)
    Display *display;
    Window w;
    unsigned long border_pixel;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
border_pixel	Specifies the entry in the color map.

The XSetWindowBorder function sets the border pixel of the window to the pixel value you specify. It uses this value as an entry into the color map to determine which color is to be used to paint the border. This can not be performed on an InputOnly window. It will cause an error to perform this on an InputOnly window.

XSetWindowBorder can generate BadMatch, BadPixmap, BadValue, and BadWindow errors.

To change and repaint a window's border tile, use XSetWindowBorderPixmap.

```
XSetWindowBorderPixmap(display, w, border_pixmap)
    Display *display;
    Window w;
    Pixmap border_pixmap;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
border_pixmap	Specifies the border pixmap. If you specify a pixmap ID, the associated pixmap is used for the border. If CopyFromParent is specified, a copy of the parent window's border pixmap is used.

The XSetWindowBorderPixmap function sets the border pixmap of the window to the pixmap you specify. It uses this entry for the border. The border Pixmap can be freed immediately if no further explicit references to it are to be made. This can not be performed on an InputOnly window. It will cause an error to perform this on an InputOnly window.

XSetWindowBorderPixmap can generate BadMatch, BadPixmap, BadValue, and BadWindow errors.

3.10. Translating Window Coordinates

Applications, mostly window managers, often need to perform a coordinate transformation from the coordinate space of one window to another window or need to determine which subwindow a coordinate lies in. XTranslateCoordinates fulfills these needs and avoids any race conditions by asking the X server to perform this operation.

```
int XTranslateCoordinates(display, src_w, dest_w, src_x, src_y, dest_x_return,
                          dest_y_return, child_return)
    Display *display;
    Window src_w, dest_w;
    int src_x, src_y;
    int *dest_x_return, *dest_y_return;
    Window *child_return;
```

display	Specifies the connection to the X server.
src_w	Specifies the window ID of the source window.
dest_w	Specifies the window ID of the destination window.
src_x *src_y*	Specify the x and y coordinates within the source window.
dest_x_return *dest_y_return*	Returns the x and y coordinates within the destination window.
child_return	Returns the child if the coordinates are contained in a mapped child of the destination window.

The XTranslateCoordinates function takes the src_x and src_y coordinates within the source window relative to the source window's origin and returns these coordinates to dest_x_return and dest_y_return relative to the destination window's origin. If XTranslateCoordinates returns zero, src_w and dest_w are on different screens, and dest_x_return and dest_y_return are zero. If the coordinates are contained in a mapped child of dest_w, that child is returned to the child argument.

XTranslateCoordinates can generate a BadWindow error.

Chapter 4

Window Information Functions

After you connect the display to the X server and create a window, you can use the Xlib window information functions to:

- Obtain information about a window
- Manipulate property lists
- Obtain and change window properties
- Manipulate window selection

4.1. Obtaining Window Information

Xlib provides functions with which you can obtain information about the window tree, the current attributes of a window, its current geometry, or the current pointer coordinates. Because they are most frequently used by window managers, these functions all return a status to indicate whether the window still exists.

To obtain a list of children, the parent, and number of children for a specified window, use XQueryTree.

```
Status XQueryTree(display, w, root_return, parent_return, children_return, nchildren_return)
    Display *display;
    Window w;
    Window *root_return;
    Window *parent_return;
    Window **children_return;
    unsigned int *nchildren_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. For this window, you obtain the list of its children, its root, its parent, and the number of children.
root_return	Returns the root window ID for the specified window.
parent_return	Returns the parent window ID for the specified window.
children_return	Returns a pointer to the list of children for the specified window.
nchildren_return	Returns the number of children for the specified window.

The XQueryTree function returns the root ID, the parent window ID, a pointer to the list of children windows, and the number of children in the list for the specified window. To free this list when it is no longer needed, use XFree. (See Section 2.4 for further information.) The children are listed in current stacking order, from bottom-most (first) to top-most (last). XQueryTree returns zero if it fails and nonzero if it succeeds.

XQueryTree can generate a BadWindow error.

To obtain the current attributes of a specified window, use XGetWindowAttributes.

```
Status XGetWindowAttributes(display, w, window_attributes_return)
    Display *display;
    Window w;
    XWindowAttributes *window_attributes_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose current attributes you want to obtain.
window_attributes_return	Returns the specified window's attributes in the XWindowAttributes structure.

The XGetWindowAttributes function returns the current attributes for the specified window to an XWindowAttributes structure. This structure is defined as follows:

```
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;                        /* back pointer to correct screen */
} XWindowAttributes;
```

The x and y members are set to the coordinates that define the location of the drawable. If the drawable is a window, these coordinates specifies the upper-left, outer- corner relative to the parent window's origin. If the drawable is a pixmap, these members are set to zero. The width and height members are set to the drawable's dimensions. For a window, these dimensions specifies the inside size of the window, not including the border.

The border_width member is set to the window's border width in pixels. If the drawable is a pixmap, this member is set to zero. The depth member is set to the depth of the pixmap (that is, bits per pixel for the object). The depth must be supported by the root of the specified drawable.

The visual member is a pointer to the screen's associated Visual structure. The root member is set to the root ID of the screen containing the window. The class member is set to the window's class and can be either InputOutput or InputOnly.

The bit_gravity member is set to the window's bit gravity and can be one of these constants:

ForgetGravity	EastGravity
NorthWestGravity	SouthWestGravity
NorthGravity	SouthGravity
NorthEastGravity	SouthEastGravity
WestGravity	StaticGravity
CenterGravity	

See the Configuring Windows Section in Chapter 3 for additional information on bit gravity.

The win_gravity member is set to the window's window gravity and can be one of these constants:

UnmapGravity	EastGravity
NorthWestGravity	SouthWestGravity
NorthGravity	SouthGravity
NorthEastGravity	SouthEastGravity
WestGravity	StaticGravity
CenterGravity	

The backing_store member is set to indicate how the X server should maintain the contents of a window. It can be set to one of the constants WhenMapped, Always, or NotUseful. The backing_planes member is set to indicate (with one bits) which bit planes of the window hold dynamic data that must be preserved in backing_stores and during save_unders. The backing_pixel member is set to indicate what values to use when restoring planes from a partial backing store.

The save_under member is set to either True or False. The colormap member is set to the colormap for the specified window and can be a colormap ID or None. The map_installed member is set to indicate whether the colormap is currently installed. It can be either True or False. The map_state member is set to indicate the state of the window and can be one of the constants IsUnmapped, IsUnviewable, or IsViewable. This member gets set to IsUnviewable if the window is mapped but some ancestor is unmapped.

The all_event_masks member is set to the bitwise inclusive OR of all event masks selected on the window by interested clients. The your_event_mask member is set to the bitwise inclusive OR of all event masks selected by the querying client. The do_not_propagate_mask member is set to the bitwise inclusive OR of the set of events that should not propagate. See Section 8.3 for a discussion of events and the event mask.

The override_redirect member is set to indicate whether this window overrides structure control facilities. It can be either True or False. Window manager clients usually should ignore the window if this member is True. Transient windows should mark which windows they are associated with. See Section 9.1.9 for further information.

The screen member is set to a screen pointer that gives you a back pointer to the correct screen. This makes it easier to obtain the screen information without having to loop over the root window fields to see which matches.

To obtain the current geometry of the specified drawable, use XGetGeometry.

```
Status XGetGeometry(display, d, root_return, x_return, y_return, width_return,
            height_return, border_width_return, depth_return)
    Display *display;
    Drawable d;
    Window *root_return;
    int *x_return, *y_return;
    unsigned int *width_return, *height_return;
    unsigned int *border_width_return;
    unsigned int *depth_return;
```

display	Specifies the connection to the X server.
d	Specifies the drawable. The drawable can be either a window or a pixmap.
root_return	Returns the root window ID for the specified window.
x_return	

y_return	Returns the x and y coordinates of the drawable. These coordinates define the location of the drawable. For a window, these coordinates specify the upper left outer corner relative to its parent's origin. For pixmaps, these coordinates are always zero.
width_return *height_return*	Returns the drawable's dimensions (width and height). For a window, these dimensions specifies the inside size, not including the border.
border_width_return	Returns the border width in pixels. The function returns the border width only if the drawable is a window. It returns zero if the drawable is a pixmap.
depth_return	Returns the depth of the pixmap (bits per pixel for the object).

The XGetGeometry function returns the root ID and the current geometry of the drawable. The geometry of the drawable includes the x and y coordinates, width and height, border width, and depth. These are described in the argument list. It is legal to pass to this function a window whose class is InputOnly.

XGetGeometry can generate a BadDrawable error.

To obtain the root window the pointer is currently on and the pointer coordinates relative to the root's origin, use XQueryPointer.

```
Bool XQueryPointer(display, w, root_return, child_return, root_x_return, root_y_return,
          win_x_return, win_y_return, mask_return)
    Display *display;
    Window w;
    Window *root_return, *child_return;
    int *root_x_return, *root_y_return;
    int *win_x_return, *win_y_return;
    unsigned int *mask_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
root_return	Returns the root window ID for the specified window. This root ID identifies the root window the pointer is currently on.
child_return	Returns the child window ID that the pointer is located in, if any.
root_x_return *root_y_return*	Returns the pointer coordinates relative to the root window's origin.
win_x_return *win_y_return*	Returns the pointer coordinates relative to the specified window.
mask_return	Returns the current state of the modifier keys and pointer buttons.

The XQueryPointer function returns the root window the pointer is logically on and the pointer coordinates relative to the root window's origin. If XQueryPointer returns False, the pointer is not on the same screen as the window associated with the window you passed to the w argument. In this case, the function returns None to child_return and zero to win_x_return and win_y_return. If XQueryPointer returns True, the pointer coordinates returned to win_x_return and win_y_return are relative to the origin of the window identified by the w argument. In this case, the function returns the ID of the child containing the pointer, if any.

The current logical state of the keyboard buttons and the modifier keys are returned in the mask_return argument. Depending on the current state of the mouse buttons and the modifier keys, XQueryPointer can set this argument to the bitwise inclusive OR of one or more of the button or modifier key bitmasks.

Note that the logical state of a device (as seen by means of the X protocol) may lag the physical state if device event processing is frozen. See Section 7.4 for further information.

XQueryPointer can generate a BadWindow error.

4.2. Properties and Atoms

A property is a collection of named typed data. Data of only one type may be associated with a single property. Clients can store and retrieve properties associated with windows. For efficiency reasons, an atom is used rather than a whole name string. XInternAtom can be used either to define new properties or to obtain the atom for new properties.

The window system has a set of predefined properties (for example, the name of a window, size hints, and so on), and users can define any other arbitrary information and can associate them with windows. Each property has a name, which is an ISO Latin-1 string. For each named property there is a unique identifier (atom) associated with it. A property also has a type, for example, string or integer. These types are also indicated using atoms, so arbitrary new types can be defined.

A property is also stored in one of several possible formats. The X server can store the information as 8-bit quantities, 16-bit quantities, or 32-bit quantities. This permits the X server to present the data in the byte order that the client expects.

Note

> If you define further properties of complex type, you must encode and decode them yourself. These functions must be carefully written if they are to be portable. See Appendix D for further information about how to write a library function.

The type of a property is defined by other properties, which allows for arbitrary extension in this type scheme.

Certain properties are predefined in the server for commonly used functions. The atoms for these properties are defined in <X11/Xatom.h>. To avoid name clashes with user symbols, the #define name for each atom has the "XA_" prefix added. See Section 4.3 for definitions of these properties. See Chapter 9 for an explanation of the functions that let you get and set much of the information stored in these predefined properties.

You can use this mechanism to communicate other information between applications. The functions described in this Section let you define new properties and get the unique Atom IDs in your applications.

Atoms occur in five distinct name spaces within the protocol.

- Selections
- Property names
- Property types
- Font properties
- Type of a ClientMessage event (none are built into the X server)

Any particular atom can have some client interpretation within each of the name spaces.

The built-in selection properties, which name properties, are:

PRIMARY
SECONDARY

The built-in property names are:

CUT_BUFFER0	RGB_GREEN_MAP
CUT_BUFFER1	RGB_RED_MAP
CUT_BUFFER2	RESOURCE_MANAGER

CUT_BUFFER3	WM_CLASS
CUT_BUFFER4	WM_CLIENT_MACHINE
CUT_BUFFER5	WM_COMMAND
CUT_BUFFER6	WM_HINTS
CUT_BUFFER7	WM_ICON_NAME
RGB_BEST_MAP	WM_ICON_SIZE
RGB_BLUE_MAP	WM_NAME
RGB_DEFAULT_MAP	WM_NORMAL_HINTS
RGB_GRAY_MAP	WM_ZOOM_HINTS
	WM_TRANSIENT_FOR

The built-in property types are:

ARC	POINT
ATOM	RGB_COLOR_MAP
BITMAP	RECTANGLE
CARDINAL	STRING
COLORMAP	VISUALID
CURSOR	WINDOW
DRAWABLE	WM_HINTS
FONT	WM_SIZE_HINTS
INTEGER	
PIXMAP	

The built-in font property types are:

MIN_SPACE	STRIKEOUT_DESCENT
NORM_SPACE	STRIKEOUT_ASCENT
MAX_SPACE	ITALIC_ANGLE
END_SPACE	X_HEIGHT
SUPERSCRIPT_X	QUAD_WIDTH
SUPERSCRIPT_Y	WEIGHT
SUBSCRIPT_X	POINT_SIZE
SUBSCRIPT_Y	RESOLUTION
UNDERLINE_POSITION	COPYRIGHT
UNDERLINE_THICKNESS	NOTICE
FONT_NAME	FAMILY_NAME
FULL_NAME	CAP_HEIGHT

Note

See Chapter 6 for further information about font property atoms.

To return an atom for a specified name, use XInternAtom.

```
Atom XInternAtom(display, atom_name, only_if_exists)
    Display *display;
    char *atom_name;
    Bool only_if_exists;
```

display	Specifies the connection to the X server.
atom_name	Specifies the name associated with the atom you want returned.

only_if_exists Specifies a boolean value that indicates whether XInternAtom creates the atom. You can pass either True or False.

The XInternAtom function returns the atom identifier associated with the string you passed to the atom_name argument. XInternAtom returns the atom for the specified atom_name if only_if_exists is True. If only_if_exists is False, the atom is created if it does not exist. Therefore, XInternAtom can return None. You should use a null-terminated ISO Latin-1 string for atom_name. Case matters: the strings "thing", "Thing", and "thinG" all designate different atoms. The atom will remain defined even after the client who defined it has gone away. It will become undefined only when the last connection to the X server closes.

XInternAtom can generate BadAlloc and BadValue errors.

To return a name for the specified atom identifier, use XGetAtomName. The definition for this function is:

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

display Specifies the connection to the X server.

atom Specifies the atom associated with the string name you want returned.

The XGetAtomName function returns the name associated with the atom identifier you passed to the atom argument. You previously obtained the atom identifier by calling XInternAtom.

XGetAtomName can generate a BadAtom error.

4.3. Obtaining and Changing Window Properties

You can attach a property list to every window. Each property has a name, a type, and a value. (See Section 4.2 for further information.) The value is an array of 8-bit, 16-bit, or 32-bit quantities, whose interpretation is left to the clients.

Xlib provides functions with which you can obtain, change, update, or interchange a window property. In addition, Xlib provides other utility functions for predefined property operations. See Chapter 9 for further information about predefined property functions.

To obtain the atom type and property format of a specified window, use XGetWindowProperty.

int XGetWindowProperty(*display*, *w*, *property*, *long_offset*, *long_length*, *delete*, *req_type*,
        actual_type_return, *actual_format_return*, *nitems_return*, *bytes_after_return*,
        prop_return)
    Display **display*;
    Window *w*;
    Atom *property*;
    long *long_offset*, *long_length*;
    Bool *delete*;
    Atom *req_type*;
    Atom **actual_type_return*;
    int **actual_format_return*;
    unsigned long **nitems_return*;
    unsigned long **bytes_after_return*;
    unsigned char ***prop_return*;

display Specifies the connection to the X server.

w Specifies the window ID. This is the window whose atom type and property format you want to obtain.

property	Specifies the property atom.
long_offset	Specifies the offset in the specified property (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 that determines whether the property is deleted from the window. You can pass one of these constants: True or False.
req_type	Specifies the atom identifier associated with the property type. You can also pass AnyPropertyType.
actual_type_return	Returns the atom identifier that defines the actual type of the property.
actual_format_return	Returns the actual format of the property.
nitems_return	Returns the actual number of 8-bit, 16-bit, or 32-bit items stored in the prop_return data.
bytes_after_return	Returns the number of bytes remaining. This is the number of bytes remaining to be read in the property if a partial read was performed.
prop_return	Returns a pointer to the data, in the specified format.

The XGetWindowProperty function returns the actual type of the property; the actual format of the property; the number of 8-bit, 16-bit, or 32-bit items transferred; the number of bytes remaining to be read in the property; and a pointer to the data actually returned. This function sets the return arguments according to the following:

- If the specified property does not exist for the specified window, XGetWindowProperty returns None to the actual_type_return argument and the value zero to the actual_format_return and bytes_after_return arguments. The nitems_return argument is empty. In this case, the delete argument is ignored.
- If the specified property exists, but its type does not match the specified type, XGetWindowProperty returns the actual property type to the actual_type_return argument; the actual property format (never zero) to the actual_format_return argument; and the property length in bytes (even if the actual_format_return is 16 or 32) to the bytes_after_return argument. It also ignores the delete argument. The nitems_return argument is empty.
- If the specified property exists, and either you assign AnyPropertyType to the req_type argument or the specified type matches the actual property type, XGetWindowProperty returns the the actual property type to the actual_type_return argument and the actual property format (never zero) to the actual_format_return argument. It also returns a value to the bytes_after_return and nitems_return arguments, by defining the following values:

  N = actual length of the stored property in bytes
     (even if the format is 16 or 32)
  I = 4 * long_offset
  T = N - I
  L = MINIMUM(T, 4 * long_length)
  A = N - (I + L)

  The returned value starts at byte index I in the property (indexing from zero), and its length in bytes is L. A BadValue error is returned if the value for long_offset causes L to be negative. The value of bytes_after_return is A, giving the number of trailing unread bytes in the stored property.

  If delete is True and bytes_after_return is zero the function deletes the property from the window and generates a PropertyNotify event on the window.

XGetWindowProperty allocates one extra byte and sets it to ASCII null, so that simple properties consisting of characters do not have to be copied into yet another string before use. The function returns Success if it executes successfully.

XGetWindowProperty can generate BadAtom, BadValue, and BadWindow errors.

To obtain the specified window's property list, use XListProperties.

```
Atom *XListProperties(display, w, num_prop_return)
    Display *display;
    Window w;
    int *num_prop_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose property list you want to obtain.
num_prop_return	Returns the length of the properties array.

The XListProperties function returns a pointer to an array of atom properties that are defined for the specified window. To free the memory allocated by this function, use XFree. (See Section 2.4 for further information.)

XListProperties can generate a BadWindow error.

To change the property of a specified window, use XChangeProperty.

```
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;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose property you want to change.
property	Specifies the property atom. The property remains defined even after the client who defined it closes its connection.
type	Specifies the type of the property. The X server does not interpret the type but simply passes it back to an application that later calls XGetProperty.
format	Specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities. This information allows the X server to correctly perform byte-swap operations as necessary. If the format is 16-bit or 32-bit, you must explicitly cast your data pointer to a (char *) in the call to XChangeProperty. Possible values are 8, 16, and 32.
mode	Specifies the mode of the operation. You can pass one of these constants: PropModeReplace, PropModePrepend, or PropModeAppend.
data	Specifies the property data.
nelements	Specifies the number of elements of the specified data format (8-bit, 16-bit, or 32-bit).

The XChangeProperty function alters the property for the specified window and causes the X server to generate a PropertyNotify event on that window. XChangeProperty does the following according to the value you assign to the mode argument:

- If the mode argument is PropModeReplace, XChangeProperty discards the previous property value.
- If the mode argument is PropModePrepend or PropModeAppend, the type and format must match the existing property value. Otherwise, XChangeProperty generates a BadMatch error. If the property is undefined, it is treated as defined with the correct type and format with zero-length data.

  For PropModePrepend, the function inserts the data before the beginning of the existing data. For PropModeAppend, the function appends the data onto the end of the existing data.

The lifetime of a property is not tied to the storing client. Properties remain until explicitly deleted, or the window is destroyed, or until the server resets. See Section 2.5 for a discussion of what happens when the connection to the X server is closed. The maximum size of a property is server dependent and, depending on the amount of memory the server has available, can vary dynamically. (If there is insufficient space, a BadAlloc error is generated.)

XChangeProperty can generate BadAlloc, BadAtom, BadMatch, BadValue, and BadWindow errors.

To rotate properties in the properties array, use XRotateWindowProperties.

XRotateWindowProperties(*display*, *w*, *properties*, *num_prop*, *npositions*)
    Display **display*;
    Window *w*;
    Atom *properties*[];
    int *num_prop*;
    int *npositions*;

display	Specifies the connection to the X server.
w	Specifies the window ID.
properties	Specifies the array of properties that are to be rotated.
num_prop	Specifies the length of the properties array.
npositions	Specifies the rotation amount.

The XRotateWindowProperties function allows you to rotate properties in the properties array and causes the X server to generate a PropertyNotify event. If the property names in the properties array are viewed as being numbered starting from zero and if there are num_prop property names in the list, then the value associated with property name I becomes the value associated with property name (I + npositions) mod N, for all I from zero to N - 1. The effect is to rotate the states by npositions places around the virtual ring of property names (right for positive npositions, left for negative npositions). If npositions mod N is nonzero, the X server generates a PropertyNotify event for each property in the order that they are listed in the array. If an atom occurs more than once in the list or no property with that name is defined for the window, a BadMatch error is generated. If a BadAtom or BadMatch error is generated, no properties are changed.

XRotateWindowProperties can generate BadAtom, BadMatch, and BadWindow errors.

To delete a property for the specified window, use XDeleteProperty.

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

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose property you want to delete.
property	Specifies the property atom.

The XDeleteProperty function deletes the specified property only if the property was defined on the specified window. XDeleteProperty causes the X server to generate a PropertyNotify event on the window, unless the property does not exist.

XDeleteProperty can generate BadAtom and BadWindow errors.

4.4. Window Selections

Selections are one method for applications to exchange data. By using the property mechanism, applications can exchange data of arbitrary types and can negotiate the type of the data. A selection can be thought of as an indirect property with a dynamic type. That is, rather than having the property stored in the X server, the property is maintained by some client (the owner). A selection is global in nature, being thought of as belonging to the user but maintained by clients, rather than being private to a particular window subhierarchy or a particular set of clients.

Xlib provides functions with which you can set, get, or request conversion of window selections. This allows applications to implement the notion of current selection, which requires notification be sent to applications when they no longer own the selection. Applications that support selection often highlight the current selection and need to be able to be informed when some other application has acquired the selection in order to be able to unhighlight the selection.

When a client asks for the contents of a selection, it specifies a selection target type. This target type can be used to control the transmitted representation of the contents. For example, if the selection is "the last thing the user clicked on," and that is currently an image, then the target type might specifies whether the contents of the image should be sent in XYFormat or ZFormat.

The target type can also be used to control the class of contents transmitted, for example, asking for the "looks" (fonts, line spacing, indentation, and so forth) of a paragraph selection, not the text of the paragraph. The target type can also be used for other purposes. The semantics are not constrained by the protocol.

To set the selection owner, use XSetSelectionOwner.

```
XSetSelectionOwner(display, selection, owner, time)
    Display *display;
    Atom selection;
    Window owner;
    Time time;
```

display	Specifies the connection to the X server.
selection	Specifies the selection atom.
owner	Specifies the owner of the specified selection atom. You can pass a window ID or None.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or CurrentTime.

The XSetSelectionOwner function changes the owner and last change time for the specified selection. The function has no effect if the value you pass to the time argument is earlier than the current last-change time of the specified selection or is later than the current X server time. Otherwise, the last-change time is set to the specified time, with CurrentTime replaced by the current server time. If the owner window is specified as None, then the owner of the selection becomes None (that is, no owner). Otherwise, the owner of the selection becomes the client executing the request.

If the new owner (whether a client or None) is not the same as the current owner of the selection, and the current owner is not None, the current owner is sent a SelectionClear event. If the client that is the owner of a selection is later terminated (that is, its connection is closed) or if the owner window it has specified in the request is later destroyed, the owner of the selection automatically reverts to None, but the last-change time is not affected. The selection atom is uninterpreted by the X server. The owner window is returned by the XGetSelectionOwner function and is reported in SelectionRequest and SelectionClear events. Selections are global to the X server.

XSetSelectionOwner can generate BadAtom and BadWindow errors.

To return selection owner, use XGetSelectionOwner.

```
Window XGetSelectionOwner(display, selection)
    Display *display;
    Atom selection;
```

display	Specifies the connection to the X server.
selection	Specifies the selection atom. This is the atom whose owner you want returned.

The XGetSelectionOwner function returns the window ID associated with the window that currently owns the specified selection. If no selection was specified, the function returns the constant None. If None is returned, there is no owner for the selection.

XGetSelectionOwner can generate a BadAtom error.

To request conversion of a selection, use XConvertSelection.

```
XConvertSelection(display, selection, target, property, requestor, time)
    Display *display;
    Atom selection, target;
    Atom property;
    Window requestor;
    Time time;
```

display	Specifies the connection to the X server.
selection	Specifies the selection atom.
target	Specifies the target atom.
property	Specifies the property atom. You also can pass None.
requestor	Specifies the requestor.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or CurrentTime.

XConvertSelection requests that the specified selection be converted to the specified target type:

- If the specified selection has an owner, the X server sends a SelectionRequest event to that owner.
- If no owner for the specified selection exists, the X server generates a SelectionNotify event to the requestor with property None. The arguments are passed on unchanged in either event.

There are two predefined selection atoms: ''PRIMARY'' and ''SECONDARY''. See Chapter 8 for more information on events, and in particular the SelectionNotify event.

XConvertSelection can generate BadAtom and BadWindow errors.

r.

Chapter 5

Graphics Resource Functions

After you connect your program to the X server by calling XOpenDisplay, you can use the Xlib graphics resource functions to:

- Create, copy, destroy, and modify colormaps
- Manipulate pixmaps
- Manipulate graphics context/state
- Use GC convenience routines

There are a number of resources used when performing graphics operations in X. Most information about performing graphics (for example, foreground color, background color, line style, and so on) are stored in resources called graphics contexts. Most graphics operations (see Chapter 6) take a graphics context or "GC" as an argument. While it, in theory, is possible to share GCs between applications, it is expected that applications will use their own GCs when performing operations, and such shared use is highly discouraged because the library may cache GC state.

Windows in X always have an associated colormap that provides a level of indirection between pixel values and color displayed on the screen. Many of the hardware displays built today have a single colormap, so the primitives are written to encourage sharing of colormap entries between applications. Because colormaps are associated with windows, X will support displays with multiple colormaps and, indeed, different types of colormaps. If there are not sufficient colormap resources in the display, some windows may not be displayed in their true colors. A window manager can set which windows are displayed in their true colors if more than one colormap is required for the color resources the applications are using.

Off screen memory or pixmaps are often used to define frequently used images for later use in graphics operations. Pixmaps are also used to define tiles or patterns for use as window backgrounds, borders, or cursors. A single bit-plane pixmap is sometimes referred to as a bitmap.

Note

> Some screens may have very limited off screen memory. You should regard it as a precious resource.

Graphics operations can be performed to either windows or pixmaps, also called drawables. Each drawable exists on a single screen and can only be used on that screen. GCs can also only be used with drawables of matching screens and depths.

5.1. Colormap Functions

Xlib provides functions with which you can manipulate a colormap. This Section discusses how to:

- Create, copy, and destroy the colormap
- Allocate and deallocate colors

5.1.1. Creating, Copying, and Destroying Colormaps

Xlib provides functions with which you can create, copy, free, or set a colormap.

The following functions manipulate the representation of color on the screen. For each possible value a pixel may take on a display, there is a color cell in the colormap. For example, if a display is 4 bits deep, pixel values 0 through 15 are defined. A colormap is the collection of the color cells. A color cell consists of a triple of red, green, and blue. As each pixel is read out of display memory, its value is taken and looked up in the colormap. The values of the cell determine what color is displayed on the screen. On a multiplane display with a black and white monitor (grayscale, but not color), these values may or may not be combined to determine the brightness on the screen.

Screens always have a default colormap. Programs will typically allocate cells out of this colormap. It is highly discouraged to write applications which monopolize color resources. On a screen that either cannot load the colormap or cannot have a fully independent colormap, only certain kinds of allocations may work. One or more (on certain hardware) colormaps may be resident (that is, installed) at one time. The XInstallColormap function (see Chapter 7) is used to install a colormap. The DefaultColormap macro returns the default colormap. The DefaultVisual macro returns the default visual type for the specified screen. Color maps are local to a particular screen. Possible visual types are represented by these constants: StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, or DirectColor. These types are more fully discussed in the Section on visual types in Chapter 3.

The introduction of color changes the view a programmer should take when dealing with a bitmap display. For example, when printing text, you write a pixel value, which is defined to be a specific color, rather than setting or clearing bits. Hardware will impose limits (number of significant bits, for example) on these values. Typically, one allocates color cells or sets of color cells. If read only, the pixel values for these colors may be shared among multiple applications, and the RGB values of the cell cannot be changed. If read/write, they are exclusively owned by the program, and the color cell associated with the pixel value may be changed at will.

The functions in this Section operate on an XColor structure:

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

The red, green and blue values are scaled between 0 and 65535. That is, on full in a color is a value of 65535 independent of the number of bit planes of the display. Half brightness in a color would be a value of 32767 and off would be 0. This representation gives uniform results for color values across displays with different numbers of bit planes.

The flags member, which can be one or more of DoRed, DoGreen, and DoBlue, is used in some functions to control which members will be set.

To create a colormap for a screen, use XCreateColormap.

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

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window on whose screen you want to create a colormap.
visual	Specifies a pointer to a visual type supported on the screen. If the visual type is not one supported by the screen, the function returns a BadMatch error.
alloc	Specifies the color map entries to be allocated. You can pass one of these constants: AllocNone or AllocAll.

The XCreateColormap function creates a color map of the specified visual type for the screen on which the specified window resides and associates the Colormap ID with it. XCreateColormap operates on a Visual structure, whose members contain information about the colormapping that is possible. Note that this does not set the colormap of the specified window, which is only used to determine the correct screen (see XSetWindowColormap).

The members of this structure pertinent to the discussion of XCreateColormap are class, red_mask, green_mask, blue_mask, and map_entries. The class member specifies the screen class and can be one of these constants: GrayScale, PseudoColor, DirectColor, StaticColor, StaticGray, or TrueColor. The red_mask, green_mask, and blue_mask members specifies the color mask values. The map_entries member specifies the number of colormap entries. The class member constant determines whether the initial values for map_entries are defined. If the class member is GrayScale, PseudoColor, or DirectColor, the initial values for map_entries are undefined. However, if the class member is StaticColor, StaticGray, or TrueColor, map_entries has initial values that are defined. These values are specific to the visual type and are not defined by the X server.

The class member constant also determines the constant you should pass to the alloc argument:

- If the class member is StaticGray, StaticColor, or TrueColor you must pass AllocNone. Otherwise, the function generates a BadMatch error.
- If the class member is any other class, you can pass AllocNone. In this case, the colormap has no values defined for map_entries. This allows your client programs to allocate the entries in the colormap. You can also pass AllocAll. In this case, XCreateColormap allocates the entire colormap as writable. The initial values of all map_entries are undefined. You cannot free any of these map_entries with a call to the function XFreeColors.

  For a colormap class of GrayScale or PseudoColor, the processing simulates a call to the function XAllocColor, where XAllocColor returns all pixel values from zero to N - 1. The value N represents the map_entries value in the specified Visual structure. For a colormap class of DirectColor, the processing simulates a call to the function XAllocColorPlanes, where XAllocColorPlanes returns a pixel value of zero and rmask, gmask, and bmask values containing the same bits as the red_mask, green_mask, and blue_mask members in the specified Visual structure.

XCreateColormap can generate BadAlloc, BadMatch, BadValue, and BadWindow errors.

To create a new colormap when allocating out of a previously shared colormap has failed due to resource exhaustion, use XCopyColormapAndFree.

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

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.

XCopyColormapAndFree:

- Creates a colormap of the same visual type and for the same screen as the cmap argument and returns the new colormap ID.
- Moves all of the client's existing allocation from cmap to the new colormap with their color values intact and their read-only or writable characteristics intact and frees those entries. Color values in other entries in the new colormap are undefined.
- 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 not created by those clients with AllocAll, the allocations to be moved are all those pixels and planes that have been allocated by the client using XAllocColor, XAllocNamedColor, XAllocColorCells, or XAllocColorPlanes and those which have not been freed since they were allocated.

XCopyColormapAndFree can generate BadAlloc and BadColor errors.

To set the colormap of a specified window, use XSetWindowColormap.

```
XSetWindowColormap(display, w, cmap)
    Display *display;
    Window w;
    Colormap cmap;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
cmap	Specifies the color map ID.

The XSetWindowColormap function sets the specified color map of the specified window.

XSetWindowColormap can generate BadColor, BadMatch, and BadWindow errors.

To delete the association between the colormap resource ID and the colormap, use XFreeColormap.

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

display	Specifies the connection to the X server.
cmap	Specifies the color map ID. This is the colormap associated with the resource ID you want to delete.

The XFreeColormap function deletes the association between the color map resource ID and the color map. However, this function has no effect on the default color map for a screen. If cmap is an installed map for a screen, it is uninstalled. See XUninstallColormap. If cmap is defined as the colormap for a window (by XCreateWindow, XSetWindowColormap, or XChangeWindowAttributes), XFreeColormap changes the colormap associated with the window to None and generates a ColormapNotify event. The colors displayed for a window with a colormap of None are not defined by X.

XFreeColormap can generate a BadColor error.

5.1.2. Allocating, Modifying, and Freeing Color Cells

Xlib provides functions with which you can allocate pixel values for colors that you need to display and can deallocate them when they are no longer needed. There are two ways of allocating color cells: explicitly as read only entries by pixel value or read/write, where you can allocate a number of color cells and planes simultaneously. The read/write cells you allocate do not have defined colors until set with XStoreColor or XStoreColors.

To determine the color names, the X server uses a color database. On a UNIX-based system, this database is /usr/lib/rgb, and a printable copy of it is stored in /usr/lib/rgb.txt. The name and contents of this file are operating system and possibly screen specific. Although you can change the values in a read/write color cell that is allocated by another application, this is considered ''anti-social'' behavior.

To allocate a read-only color cell, use XAllocColor.

```
Status XAllocColor(display, cmap, screen_in_out)
    Display *display;
    Colormap cmap;
    XColor *screen_in_out;
```

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
screen_in_out	Specifies or returns the values actually used in the color map.

The XAllocColor function allocates a read-only color map entry corresponding to the closest red, green, and blue values supported by the hardware. XAllocColor returns the pixel value of the color closest to the specified RGB elements supported by the hardware and returns the red, green, and blue values actually used. The corresponding colormap cell is read-only. In addition, XAllocColor returns zero if there was a problem (typically lack of resources) or nonzero if it succeeded. Read-only colormap cells are shared among clients. When the last client deallocates a shared cell, it is deallocated.

XAllocColor can generate BadAlloc and BadColor errors.

To allocate a read-only color cell by name and return the closest color supported by the hardware, use XAllocNamedColor.

```
Status XAllocNamedColor(display, cmap, color_name, visual_def_return, exact_def_return)
    Display *display;
    Colormap cmap;
    char *color_name;
    XColor *visual_def_return, *exact_def_return;
```

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
color_name	Specifies the color name string (for example, ''red'') whose color definition structure you want returned.
visual_def_return	Returns the closest RGB values provided by the hardware.
exact_def_return	Returns the exact RGB values.

The XAllocNamedColor function looks up the named color with respect to the screen that is associated with the specified color map. Both the exact data base definition and the closest color supported by the screen are returned. The allocated color cell is read-only. You should use the ISO Latin-1 encoding, and upper/lower case does not matter.

XAllocNamedColor can generate BadAlloc, BadColor, and BadName errors.

To look up the name of a color, use XLookupColor.

```
Status XLookupColor(display, cmap, color_name, visual_def_return, exact_def_return)
    Display *display;
    Colormap cmap;
    char *color_name;
    XColor *visual_def_return, *exact_def_return;
```

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
color_name	Specifies the color name string (for example, ''red'') whose color definition structure you want returned.
visual_def_return	Returns the closest RGB values provided by the hardware.

*exact_def_return*Returns the exact RGB values.

The XLookupColor function looks up the string name of a color with respect to the screen associated with the specified cmap and returns both the exact color values and the closest values provided by the screen with respect to the visual type of the specified cmap. You should use the ISO Latin-1 encoding for the name, and upper/lower case does not matter. XLookupColor returns nonzero if the spec existed in the RGB database or zero if it did not exist.

To allocate read/write color cell and color plane combinations for a PseudoColor model, use XAllocColorCells.

Status XAllocColorCells(*display*, *cmap*, *contig*, *plane_masks_return*, *nplanes*,
pixels_return, *ncolors*)
Display **display*;
Colormap *cmap*;
Bool *contig*;
unsigned long *plane_masks_return*[ ];
unsigned int *nplanes*;
unsigned long *pixels_return*[ ];
unsigned int *ncolors*;

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
contig	Specifies a boolean value. You pass the value 1 if the planes must be contiguous or the value 0 if the planes do not need to be contiguous.
plane_mask_return	Returns an array of plane masks.
nplanes	Specifies the number of plane masks that are to be returned in the plane masks array.
pixels_return	Returns an array of pixel values.
ncolors	Specifies the number of pixel values that are to be returned in the pixels_return array.

The XAllocColorCells function allocates read/write color cells. The number of colors must be positive and the number of planes nonnegative. Otherwise, it generates a BadValue error. If ncolors and nplanes are requested, then ncolors pixels and nplane 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 pixel with zero or more masks, ncolors* $2^{nplanes}$ distinct pixels can be produced. All of these are allocated writable by the request. For GrayScale or PseudoColor, each mask will have exactly one bit, and, for DirectColor, each will have exactly three bits. If contigs is True, and if all 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.

XAllocColorCells can generate BadAlloc, BadColor, and BadValue errors.

To allocate read/write color resources for DirectColor visual types, use XAllocColorPlanes.

Status XAllocColorPlanes(*display*, *cmap*, *contig*, *pixels_return*, *ncolors*, *nreds*, *ngreens*, *nblues*, *rmask_return*, *gmask_return*, *bmask_return*)
    Display **display*;
    Colormap *cmap*;
    Bool *contig*;
    unsigned long *pixels_return*[ ];
    int *ncolors*;
    int *nreds*, *ngreens*, *nblues*;
    unsigned long **rmask_return*, **gmask_return*, **bmask_return*;

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
contig	Specifies a boolean value. You pass the value 1 if the planes must be contiguous or the value 0 if the planes do not need to be contiguous.
pixels_return	Returns an array of pixel values. XAllocColorPlanes returns the pixel values in this array.
ncolors	Specifies the number of pixel values that are to be returned in the pixels_return array.
nreds *ngreens* *nblues*	Specify the number of red, green, and blue colors (shades). The value you pass must be non-negative.
rmask_return *gmask_return* *bmask_return*	Return bit masks for the red, green, and blue planes.

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

XAllocColorPlanes can generate BadAlloc, BadColor, and BadValue errors.

To store RGB values into colormap cells, use XStoreColors.

XStoreColors(*display*, *cmap*, *color*, *ncolors*)
    Display **display*;
    Colormap *cmap*;
    XColor *color*[ ];
    int *ncolors*;

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
color	Specifies an array of color definition structures to be stored.

ncolors	Specifies the number of XColor structures in the color definition array.

The XStoreColors function changes the color map entries of the pixel values specified in the pixel members of the XColor structures. You specifies which color components are to be changed by passing DoRed, DoGreen, and/or DoBlue to the flags members of the XColor structures. If the colormap is an installed map for its screen, the changes are visible immediately. XStoreColors changes the specified pixels if they are allocated writable in cmap by any client, even if one or more pixels generates an error. A BadValue error is generated if a specified pixel is not a valid index into cmap and a BadAccess error is generated if a specified pixel either is unallocated or is allocated read-only. If more than one pixel is in error, it is arbitrary as to which one is reported.

XStoreColors can generate BadAccess, BadColor, and BadValue errors.

To store an RGB value into a single colormap cell, use XStoreColor.

```
XStoreColor(display, cmap, color)
    Display *display;
    Colormap cmap;
    XColor *color;
```

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
color	Specifies the pixel and RGB values.

The XStoreColor function changes the color map entry of the pixel value specified in the pixel member of the XColor structure. XStoreColor changes the:

- Color map entry of the pixel value. You specified this value in the pixel member of the XColor structure. This pixel value must be a read/write cell and a valid index into cmap. A BadValue error is generated if a specified pixel is not a valid index into cmap.
- Red, green, and/or blue color components. You specifies which color components to be changed by passing DoRed, DoGreen, and/or DoBlue to the flags member of the XColor structure. If the colormap is an installed map for its screen, the changes are visible immediately.
- Specified pixel if it is allocated writable in cmap by any client, even if the pixel generates an error.

XStoreColor can generate BadColor and BadValue errors.

To set the color of a pixel to the named color, use XStoreNamedColor.

```
XStoreNamedColor(display, cmap, color, pixel, flags)
    Display *display;
    Colormap cmap;
    char *color;
    unsigned long pixel;
    int flags;
```

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
color	Specifies the color name string (for example, "red"). The function then allocates this color cell. You should use the ISO Latin-1 encoding, and upper/lower case does not matter.
pixel	Specifies the entry in the color map.
flags	Specifies which red, green, and blue indexes are set.

The XStoreNamedColor function looks up the named color with respect to the screen associated with cmap and stores the result in cmap. The pixel argument determines the entry in the colormap. The flags argument determines which of the red, green, and blue indexes are set. You can set this member to the bitwise inclusive OR of the bits DoRed, DoGreen, and DoBlue. A BadValue error is generated if a specified pixel is not a valid index into cmap and a BadAccess error is generated if a specified pixel either is unallocated or is allocated read-only. If more than one pixel is in error, it is arbitrary as to which one is reported.

XStoreNamedColor can generate BadAccess, BadColor, BadName, and BadValue errors.

To free colormap cells, use XFreeColors.

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

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
pixels	Specifies an array of pixel values. These pixel values map to the cells in the specified colormap.
npixels	Specifies the number of pixels.
planes	Specifies the planes you want to free.

The XFreeColors function frees the cells represented by pixels whose values are in the pixels array. The planes argument should not have any bits in common with any of the pixels. The set of all pixels is produced by ORing together subsets of the planes argument with the pixels. The request frees all of these pixels that were allocated by the client (using XAllocColor, XAllocNamedColor, XAllocColorCells, and XAllocColorPlanes). Note that freeing an individual pixel obtained from XAllocColorPlanes may not actually allow it to be reused until all of its related pixels are also freed.

All specified pixels that are allocated by the client in cmap are freed, even if one or more pixels produce an error. A BadValue error is generated if a specified pixel is not a valid index into cmap. A BadAccess error is generated if a specified pixel is not allocated by the client (that is, is unallocated or is only allocated by another client). If more than one pixel is in error, the one reported is arbitrary.

XFreeColors can generate BadAccess, BadColor, and BadValue errors.

5.1.3. Reading Entries in a Colormap

The XQueryColor and XQueryColors functions returns the red, green, and blue color values stored in the specified color map for the pixel value you pass in the pixel member of the XColor structure(s). The values returned for an unallocated entry are undefined. They also set the flags member in the XColor structure to all three colors. A BadValue error is generated if a pixel is not a valid index into the specified colormap. If more than one pixel is in error, it is arbitrary as to which one is reported.

To query the RGB values of a single specified pixel value, use XQueryColor.

```
XQueryColor(display, cmap, def_return)
    Display *display;
    Colormap cmap;
    XColor *def_return;
```

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
def_in_out	Specifies or returns the RGB values for the pixel specified in the structure.

XQueryColor can generate BadColor and BadValue errors.

To query the RGB values of an array of pixels stored in color structures, use XQueryColors.

```
XQueryColors(display, cmap, defs_return, ncolors)
    Display *display;
    Colormap cmap;
    XColor defs_return[];
    int ncolors;
```

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
defs_in_out	Specifies or returns an array of color definition structures.
ncolors	Specifies the number of XColor structures in the color definition array.

XQueryColors can generate BadColor and BadValue errors.

For these functions, the RGB values for each pixel in the XColor structures are returned, and the DoRed, DoGreen, and DoBlue flags are set in the flags member.

5.2. Creating and Freeing Pixmaps

Xlib provides functions with which you can create or free a pixmap. Pixmaps can only be used on the screen on which they were created. Pixmaps are off-screen resources that are used for a number of operations. A bitmap is a single bit pixmap. These include defining cursors as tiling patterns or as the source for certain raster operations. Most graphics requests can operate either on a window or on a pixmap.

To create a pixmap of a specified size, use XCreatePixmap.

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

display	Specifies the connection to the X server.
d	Specifies which screen the pixmap is created on.
width *height*	Specify the width and height. These dimensions define the width and height of the pixmap. The values you pass must be nonzero.
depth	Specifies the depth of the pixmap. The depth must be supported by the root of the specified drawable.

The XCreatePixmap function creates a pixmap of the width, height, and depth you specified. It also assigns the pixmap ID to it. It is valid to pass a window whose class is InputOnly to the drawable argument. The width and height arguments must be nonzero. Otherwise, XCreatePixmap generates a BadValue error. The depth argument must be one of the depths supported by the root window of the specified drawable. Otherwise, it generates a BadValue error.

The server uses the drawable argument to determine which screen the pixmap is stored on. The pixmap can only be used on this screen and only with other drawables of the same depth. (See XCopyPlane for an exception to this rule). The initial contents of the pixmap are undefined. If this routine returns zero, there was insufficient space for the pixmap.

XCreatePixmap can generate BadAlloc, BadDrawable, and BadValue errors.

To free all storage associated with a specified pixmap, use XFreePixmap.

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

display	Specifies the connection to the X server.
pixmap	Specifies the pixmap.

The XFreePixmap function first deletes the association between the pixmap ID and the pixmap. Then, the X server frees the pixmap storage when no other resources reference it. The pixmap should never be referenced again.

XFreePixmap can generate a BadPixmap error.

5.3. Manipulating Graphics Context/State

Most attributes of graphics operations are stored in Graphic Contexts or GCs. These include line width, line style, plane mask, foreground, background, tile, stipple, clipping region, end style, join style, and so on. Graphics operations (for example, drawing lines) use these values to determine the actual drawing operation. Extensions to X may add additional components to GCs. Xlib provides calls for changing the state of GCs.

Xlib implements a write-back cache for all elements of a GC that are not resource IDs to allow it to implement the transparent coalescing of changes to GCs. For example, a call to XSetForeground of a GC followed by a call to XSetLineAttributes will result in only a single change GC protocol request to the server. GCs are neither expected nor encouraged to be shared between client applications, so this write-back caching should present no problems. Applications cannot share GCs without external synchronization. Therefore, sharing GCs between applications is highly discouraged.

The specified components of the new graphics context in valuemask_create are set to the values passed in the values argument. The other values defaulto the following:

Component	Value
function:	GXcopy
plane_mask:	All ones
foreground:	0
background:	1
line_width:	0
line_style:	LineSolid
cap_style:	CapButt
join_style:	JoinMiter
fill_style:	FillSolid
fill_rule:	EvenOddRule
arc_mode:	ArcPieSlice
tile:	Pixmap of unspecified size filled with foreground pixel (that is, client specified pixel if any, else 0) (subsequent changes to foreground do not affect this pixmap)
stipple:	Pixmap of unspecified size filled with ones
ts_x_origin:	0
ts_y_origin:	0
font:	<implementation dependent>
subwindow_mode:	ClipByChildren

Component	Value
graphics_exposures:	True
clip_x_origin:	0
clip_y_origin:	0
clip_mask:	None
dash_offset:	0
dashes:	4 (that is, the list [4, 4])

Note that foreground and background are not set to any values likely to be useful on a color display.

You use display functions when you update a Section of the screen (destination) with bits from somewhere else (source). Many GC functions take one of these display functions as an argument. The function defines how the new destination bits are to be computed from the source bits and the old destination bits. GXcopy is typically the most useful because it will work on a color display, but special applications may use other functions, particularly in concert with particular planes of a color display. The 16 such functions, defined in <X11/X.h>, are:

Function Name	Hex Code	Operation
GXclear	0x0	0
GXand	0x1	src AND dst
GXandReverse	0x2	src AND NOT dst
GXcopy	0x3	src
GXandInverted	0x4	(NOT src) AND dst
GXnoop	0x5	dst
GXxor	0x6	src XOR dst
GXor	0x7	src OR dst
GXnor	0x8	(NOT src) AND (NOT dst)
GXequiv	0x9	(NOT src) XOR dst
GXinvert	0xa	NOT dst
GXorReverse	0xb	src OR (NOT dst)
GXcopyInverted	0xc	NOT src
GXorInverted	0xd	(NOT src) OR dst
GXnand	0xe	(NOT src) OR (NOT dst)
GXset	0xf	1

Many of the color functions below take either pixel values or planes as an argument. The planes argument is of type long and it specifies which planes of the display are to be modified, one bit per plane. A monochrome display has only one plane and will be the least significant bit of the word. As planes are added to the display hardware, they will occupy more significant bits in the plane mask.

A macro constant AllPlanes can be used to refer to all planes of a display simultaneously (~0). Most operations use an object called a GC, which is short for Graphics Context. The contents of the GC object are private to the library. Several procedures take structures of type GCValues. The following lists each entry by its defined value, not by its position in the XGCValues structure:

```
#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)

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

In graphics operations, given a source and destination pixel, the result is computed bitwise on corresponding bits of the pixels. That is, a Boolean operation is performed in each bit plane. The plane_mask restricts the operation to a subset of planes. That is, the result is computed by the following:

((src FUNC dst) AND plane-mask) OR (dst AND (NOT plane-mask))

Range checking is not performed on the values for foreground, background, or plane_mask. They are simply truncated to the appropriate number of bits. The line_width is measured in pixels and either can be greater than or equal to one (wide line) or can be the special value zero (thin line).

Wide lines are drawn centered on the path described by the graphics request. Unless otherwise specified by the join or cap style, the bounding box of a wide line with endpoints [x1, y1], [x2, y2], and width w is a rectangle with vertices at the following real coordinates:

[x1-(w*sn/2), y1+(w*cs/2)], [x1+(w*sn/2), y1-(w*cs/2)],
[x2-(w*sn/2), y2+(w*cs/2)], [x2+(w*sn/2), y2-(w*cs/2)]

The sn is the sine of the angle of the line and cs is the cosine of the angle of the line. A pixel is part of the line and, hence, is drawn, if the center of the pixel is fully inside the bounding box (which is viewed as having infinitely thin edges). If the center of the pixel is exactly on the bounding box, it is part of the line if and only if the interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are part of the line if and only if the interior is immediately below (y increasing direction).

Thin lines (zero line_width) are one pixel wide lines drawn using an unspecified, device dependent algorithm. There are only two constraints on this algorithm.

1. If a line is drawn unclipped from [x1,y1] to [x2,y2] and if another line is drawn unclipped from [x1+dx,y1+dy] to [x2+dx,y2+dy], a point [x,y] is touched by drawing the first line if and only if the point [x+dx,y+dy] is touched by drawing the second line.
2. The effective set of points comprising a line cannot be affected by clipping. That is, a point is touched in a clipped line if and only if the point lies inside the clipping region and the point would be touched by the line when drawn unclipped.

A wide line drawn from [x1,y1] to [x2,y2] always draws the same pixels as a wide line drawn from [x2,y2] to [x1,y1], not counting cap and join styles. Implementors are encouraged to make this property true for thin lines, but it is not required. A line_width of zero may differ from a line_width of one in which pixels are drawn. This permits the use of many manufacturer's line drawing hardware, which may run many times faster than than the more precisely specified wide lines.

In general, drawing a thin line will be faster than drawing a wide line of width one. However, because of their different drawing algorithms, thin lines may not mix well, aesthetically speaking, with wide lines. If it is desirable to obtain precise and uniform results across all displays, a client should always use a line_width of one, rather than a line_width of zero.

The line-style defines which Sections of a line are drawn:

LineSolid	The full path of the line is drawn.
LineDoubleDash	The full path of the line is drawn, but the even dashes are filled differently than the odd dashes (see fill-style) with CapButt style used where even and odd dashes meet.
LineOnOffDash	Only the even dashes are drawn, and cap-style applies to all internal ends of the individual dashes, except CapNotLast is treated as CapButt.

The cap_style defines how the endpoints of a path are drawn:

CapNotLast	Equivalent to CapButt, except that for a line_width of zero or one the final endpoint is not drawn.
CapButt	Square at the endpoint (perpendicular to the slope of the line) with no projection beyond.
CapRound	A circular arc with the diameter equal to the line_width, centered on the endpoint. (This is equivalent to CapButt for line_width zero or one).
CapProjecting	Square at the end, but the path continues beyond the endpoint for a distance equal to half the line_width. (This is equivalent to CapButt for line_width zero or one).

The join_style defines how corners are drawn for wide lines:

JoinMiter	The outer edges of two lines extend to meet at an angle.
JoinRound	A circular arc with diameter equal to the line_width, centered on the joinpoint.
JoinBevel	CapButt endpoint styles, and then the triangular notch filled.

For a line with coincident endpoints (x1=x2, y1=y2), when the cap_style is applied to both endpoints, the semantics depends on the line_width and the cap_style:

CapNotLast	thin	Device dependent, but the desired effect is that nothing is drawn.
CapButt	thin	Device dependent, but the desired effect is that a single pixel is drawn.
CapRound	thin	Same as CapButt/thin.
CapProjecting	thin	Same as Butt/thin.
CapButt	wide	Nothing is drawn.
CapRound	wide	The closed path is a circle, centered at the endpoint, with diameter equal to the line_width.
CapProjecting	wide	The closed path is a square, aligned with the coordinate axes, centered at the endpoint, with sides equal to the line_width

For a line with coincident endpoints (x1=x2, y1=y2), when the join_style is applied at one or both endpoints, the effect is as if the line was removed from the overall path. However, if the total path consists of or is reduced to a single point joined with itself, the effect is the same as when the cap_style is applied at both endpoints.

The tile/stipple and clip origins are interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The tile pixmap must have the same root and depth as the graphics context. Otherwise, the function generates a BadMatch error. The stipple pixmap must have depth one and must have the same root as the graphics context (else a BadMatch error). For stipple operations where the fill_style' is FillStippled but not FillOpaqueStippled, the stipple pattern is tiled in a single plane and acts as an additional clip mask to be ANDed with the clip_mask. Any size pixmap can be used for tiling or stippling, although some sizes may be faster to use than others.

The fill_style defines the contents of the source for line, text, and fill requests. For all text and fill requests (for example, XDrawText, XDrawText16, XFillRectangle, XFillPolygon, and XFillArc); for line requests (for example, XDrawLine, XDrawSegments, XDrawRectangle, XDrawArc) with line_style LineSolid; and for the even dashes for line requests with line_style LineOnOffDash or LineDoubleDash the following apply:

FillSolid	Foreground.
FillTiled	Tile.
FillOpaqueStippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one.
FillStippled	Foreground masked by stipple.

When drawing lines with line_style LineDoubleDash, the odd dashes are controlled by the fill_style in the following manner:

FillSolid	Background.

FillTiled	Same as for even dashes.
FillOpaqueStippled	Same as for even dashes.
FillStippled	Background masked by stipple.

Storing a pixmap in a graphics context might or might not result in a copy being made. If the pixmap is later used as the destination for a graphics request, the change might or might not be reflected in the graphics context. If the pixmap is used simultaneously in a graphics request both as a destination and as a tile or stipple, the results are not defined.

For optimum performance, you should draw as much as possible with the same GC (without changing its components). The costs of changing GC components relative to using different GCs depend upon the display hardware and the server implementation.

It is quite likely that some amount of graphics context information will be cached in display hardware and that such hardware can only cache a small number of graphics contexts.

The dash_list value allowed here is actually a simplified form of the more general patterns that can be set with XSetDashes. Specifies

ing a value of N here is equivalent to specifiesing the two element list [N, N] in XSetDashes. The value must be nonzero. Otherwise, the function generates a BadValue error. The meaning of dash_offset and dash_list are explained for the XSetDashes function.

The clip_mask restricts writes to the destination drawable. If a pixmap is specified as the clip_mask, it must have depth one and have the same root as the graphics context. Otherwise, the function generates a BadMatch error. If clip_mask is None, the pixels are always drawn, regardless of the clip origin. The clip_mask can also be set with the XSetClipRectangles request. Only pixels where the clip-mask has a one bit are drawn. Pixels are not drawn outside the area covered by the clip_mask or where the clip_mask has a zero bit. It affects all graphics requests. The clip_mask does not clip sources. The clip_mask origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

For ClipByChildren, both source and destination windows are additionally clipped by all viewable InputOutput children. For IncludeInferiors, neither source nor destination window is clipped by inferiors. This will result in including subwindow contents in the source and drawing through subwindow boundaries of the destination. The use of IncludeInferiors on a window of one depth with mapped inferiors of differing depth is not illegal, but the semantics are undefined by the core protocol.

The fill_rule defines what pixels are inside (drawn) for paths given in XFillPolygon requests. EvenOddRule means a point is inside if an infinite ray with the point as origin crosses the path an odd number of times.

For WindingRule, a point is inside if an infinite ray with the point as origin crosses an unequal number of clockwise and counterclockwise directed path segments. A clockwise directed path segment is one which crosses the ray from left to right as observed from the point. A counterclockwise segment is one which crosses the ray from right to left as observed from the point. The case where a directed line segment is coincident with the ray is uninteresting because you can simply choose a different ray that is not coincident with a segment.

For both EvenOddRule and WindingRule, a point is infinitely small, and the path is an infinitely thin line. A pixel is inside if the center point of the pixel is inside, and the center point is not on the boundary. If the center point is on the boundary, the pixel is inside if and only if the polygon interior is immediately to its right (x increasing direction). Pixels with centers along a horizontal edge are a special case and are inside if and only if the polygon interior is immediately below (y increasing direction).

The arc_mode controls filling in the XFillArcs function and can be either ArcPieSlice or ArcChord. The graphics_exposure flag controls GraphicsExpose event generation for XCopyArea and XCopyPlane requests (and any similar requests defined by extensions).

To create a new graphics context that is usable with the specified drawable, use XCreateGC.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
valuemask_create	Specifies which components in the graphics context are to be set using information in the XGCValues structure. This argument is the bitwise inclusive OR of one or more of the valid GC component masks.
values	Specifies a pointer to the XGCValues structure.

The XCreateGC function creates a graphics context and returns a GC. The graphics context can be used with any destination drawable having the same root and depth as the specified drawable. Use with other drawables results in a BadMatch error.

XCreateGC can generate BadAlloc, BadDrawable, BadFont, BadMatch, BadPixmap, and BadValue errors.

To copy components from a source graphics context to a destination graphics context, use XCopyGC.

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

display	Specifies the connection to the X server.
src	Specifies the components of the source graphics context.
valuemask_copy	Specifies which components in the source graphics context are to be copied to the destination graphics context. This argument is the bitwise inclusive OR of one or more of the valid GC component masks.
dest	Specifies the destination graphics context.

The XCopyGC function copies the specified components from the source graphics context to the destination graphics context. The source and destination graphics contexts must have the same root and depth. Otherwise, XCopyGC generates a BadMatch error. The valuemask_copy specifies which component to copy, as for XCreateGC.

XCopyGC can generate BadAlloc, BadGC, BadMatch, and BadValue errors.

To change the components in the specified graphics context, use XChangeGC.

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

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
valuemask_change	Specifies which components in the graphics context are to be changed using information in the XGCValues structure. This argument is the bitwise inclusive

	OR of one or more of the valid GC component masks.
values	Specifies a pointer to the XGCValues structure.

The XChangeGC function changes the components specified by the valuemask_change argument for the specified graphics context. The values argument contains the values to be set. The values and restrictions are the same as for XCreateGC. Changing the clip_mask also overrides any previous XSetClipRectangles request on the context. Changing the dash_offset or dash_list overrides any previous XSetDashes request on the context. The order in which components are verified and altered is server-dependent. If an error is generated, a subset of the components may have been altered.

XChangeGC can generate BadAlloc, BadFont, BadGC, BadMatch, BadPixmap, and BadValue errors.

To free the specified graphics context, use XFreeGC.

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

display	Specifies the connection to the X server.
gc	Specifies the graphics context.

The XFreeGC function destroys the specified graphics context as well as the shadow copy.

XFreeGC can generate a BadGC error.

5.4. Using GC Convenience Routines

This Section discusses how to set the:

- Foreground, background, plane mask, or function components
- Line attributes and dashes components
- Fill style and fill rule components
- Fill tile and stipple components
- Font component
- Clip region component
- Arc mode, subwindow mode, and graphics exposure components

5.4.1. Setting the Foreground, Background, Plane Mask, or Function

To set the foreground, background, plane mask, and function components for the specified graphics context, use XSetState.

```
XSetState(display, gc, foreground, background, function, plane_mask)
    Display *display;
    GC gc;
    unsigned long foreground, background;
    int function;
    unsigned long plane_mask;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
foreground	Specifies the foreground you want to set for the specified graphics context.
background	Specifies the background you want to set for the specified graphics context.
function	Specifies the function you want to set for the specified graphics context.
plane_mask	Specifies the plane mask.

XSetState can generate BadGC and BadValue errors.

To set the display function in the specified graphics context, use XSetFunction.

```
XSetFunction(display, gc, function)
    Display *display;
    GC gc;
    int function;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
function	Specifies the function you want to set for the specified graphics context.

XSetFunction can generate BadGC and BadValue errors.

To set the plane mask of the specified graphics context, use XSetPlaneMask.

```
XSetPlaneMask(display, gc, plane_mask)
    Display *display;
    GC gc;
    unsigned long plane_mask;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
plane_mask	Specifies the plane mask.

XSetPlaneMask can generate a BadGC error.

To set the foreground of the specified graphics context, use XSetForeground.

```
XSetForeground(display, gc, foreground)
    Display *display;
    GC gc;
    unsigned long foreground;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
foreground	Specifies the foreground you want to set for the specified graphics context.

XSetForeground can generate a BadGC error.

To set the background of the specified graphics context, use XSetBackground.

```
XSetBackground(display, gc, background)
    Display *display;
    GC gc;
    unsigned long background;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
background	Specifies the background you want to set for the specified graphics context.

XSetBackground can generate a BadGC error.

5.4.2. Setting the Line Attributes and Dashes

To set the line drawing components of the specified graphics context, use XSetLineAttributes.

```
XSetLineAttributes(display, gc, line_width, line_style, cap_style, join_style)
    Display *display;
    GC gc;
    unsigned int line_width;
    int line_style;
    int cap_style;
    int join_style;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
line_width	Specifies the line width you want to set for the specified graphics context.
line_style	Specifies the line style you want to set for the specified graphics context. Possible values are LineSolid (solid), LineOnOffDash (on-off dash), or LineDoubleDash (double dash).
cap_style	Specifies the line and cap style you want to set for the specified graphics context. Possible values are CapNotLast, CapButt, CapRound, or CapProjecting.
join_style	Specifies the line-join style you want to set for the specified graphics context. Possible values are JoinMiter, JoinRound, or JoinBevel.

XSetLineAttributes can generate BadGC and BadValue errors.

To set the dash_offset and dash_list for dashed line styles of the specified graphics context, use XSetDashes.

```
XSetDashes(display, gc, dash_offset, dash_list, n)
     Display *display;
     GC gc;
     int dash_offset;
     char dash_list[];
     int n;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
dash_offset	Specifies the phase of the pattern for the dashed line style you want to set for the specified graphics context.
dash_list	Specifies the dash list for the dashed line style you want to set for the specified graphics context.
n	Specifies the number of elements in the dash list argument.

The XSetDashes function sets the dash_offset and dash_list for dashed line styles in the specified graphics context. There must be at least one element in the specified dash_list. Otherwise, XSetDashes generates a BadValue error. The initial and alternating elements (2nd, 4th, and so on) of the dash_list are the even dashes, while the others are the odd dashes. All of the elements must be nonzero. Otherwise, it generates a BadValue error. Specifies ing an odd-length list is equivalent to specifiesing the same list concatenated with itself to produce an even-length list.

The dash_offset defines the phase of the pattern, specifiesing how many elements into the dash_list the pattern should actually begin in any single graphics request. Dashing is continuous through path elements combined with a join_style, but is reset to the dash_offset each time a cap_style is applied at a line endpoint.

The unit of measure for dashes is the same as in the ordinary coordinate system. Ideally, a dash length is measured along the slope of the line, but implementations are only required to match this ideal for horizontal and vertical lines. Failing the ideal semantics, it is suggested that the length be measured along the major axis of the line. The major axis is defined as the x axis for lines drawn at an angle of between –45 and +45 degrees or between 315 and 225 degrees from the x axis. For all other lines, the major axis is the y axis. The default dash_list in a newly created GC is equivalent to [4,4].

XSetDashes can generate BadAlloc, BadGC, and BadValue errors.

5.4.3. Setting the Fill Style and File Rule

To set the fill style of the specified graphics context, use XSetFillStyle.

```
XSetFillStyle(display, gc, fill_style)
    Display *display;
    GC gc;
    int fill_style;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
fill_style	Specifies the fill style you want to set for the specified graphics context. Possible values are FillSolid, FillTiled, FillStippled, or FillOpaqueStippled.

XSetFillStyle can generate BadGC and BadValue errors.

To set the fill rule of the specified graphics context, use XSetFillRule.

```
XSetFillRule(display, gc, fill_rule)
    Display *display;
    GC gc;
    int fill_rule;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
fill_rule	Specifies the fill rule you want to set for the specified graphics context. You can pass one of these constants: EvenOddRule or WindingRule.

XSetFillRule can generate BadGC and BadValue errors.

5.4.4. Setting the Fill Tile and Stipple

Some displays have hardware support for tiling or stippling with patterns of specific sizes. Tiling and stippling operations that restrict themselves to those sizes run much faster than such operations with arbitrary size patterns. Xlib provides functions with which you can determine the best size, tile, or stipple for the display as well as set the tile or stipple shape and the tile/stipple origin.

To obtain the best size of a tile, stipple, or cursor, use XQueryBestSize.

```
Status XQueryBestSize(display, class, which_screen, width, height, width_return, height_return)
    Display *display;
    int class;
    Drawable which_screen;
    unsigned int width, height;
    unsigned int *width_return, *height_return;
```

display	Specifies the connection to the X server.
class	Specifies the class that you are interested in. You can pass one of these constants: TileShape, CursorShape, or StippleShape.
which_screen	Specifies any drawable on a screen.
width	

height Specify the width and height.

width_return
height_return Returns the width and height of the object best supported by the display hardware.

The XQueryBestSize function returns the best or closest size to the specified size. For CursorShape, this is the largest size that can be fully displayed on the screen specified by which_screen. For TileShape, this is the size that can be tiled fastest. For StippleShape, this is the size that can be stippled fastest. For CursorShape, the drawable indicates the desired screen. For TileShape and StippleShape, the drawable indicates the screen and possibly the window class and depth. An InputOnly window cannot be used as the drawable for TileShape or StippleShape. Otherwise, XQueryBestSize generates a BadMatch error.

XQueryBestSize can generate BadDrawable, BadMatch, and BadValue errors.

To obtain the best fill tile shape, use XQueryBestTile.

Status XQueryBestTile(*display*, *which_screen*, *width*, *height*, *width_return*, *height_return*)
    Display **display*;
    Drawable *which_screen*;
    unsigned int *width*, *height*;
    unsigned int **width_return*, **height_return*;

display Specifies the connection to the X server.

which_screen Specifies any drawable on a screen.

width
height Specify the width and height.

width_return
height_return Returns the width and height of the object best supported by the display hardware.

The XQueryBestTile function returns the best or closest size, that is, the size that can be tiled fastest on the screen specified by which_screen. The drawable indicates the screen and possibly the window class and depth. An InputOnly window cannot be used as the drawable for XQueryBestTile. Otherwise, it generates a BadMatch error.

XQueryBestTile can generate BadDrawable and BadMatch errors.

To obtain the best stipple shape, use XQueryBestStipple.

Status XQueryBestStipple(*display*, *which_screen*, *width*, *height*, *width_return*, *height_return*)
    Display **display*;
    Drawable *which_screen*;
    unsigned int *width*, *height*;
    unsigned int **width_return*, **height_return*;

display Specifies the connection to the X server.

which_screen Specifies any drawable on a screen.

width
height Specify the width and height.

width_return
height_return Returns the width and height of the object best supported by the display hardware.

The XQueryBestStipple function returns the best or closest size, that is, the size that can be stippled fastest on the screen specified by which_screen. The drawable indicates the screen and possibly the window class and depth. An InputOnly window cannot be used as the drawable for XQueryBestStipple. Otherwise, it generates a BadMatch error.

XQueryBestStipple can generate BadDrawable and BadMatch errors.

To set the fill tile of the specified graphics context, use XSetTile.

```
XSetTile(display, gc, tile)
    Display *display;
    GC gc;
    Pixmap tile;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
tile	Specifies the fill tile you want to set for the specified graphics context.

XSetTile can generate BadAlloc, BadGC, BadMatch, and BadPixmap errors.

To set the stipple of the specified graphics context, use XSetStipple.

```
XSetStipple(display, gc, stipple)
    Display *display;
    GC gc;
    Pixmap stipple;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
stipple	Specifies the stipple you want to set for the specified graphics context.

XSetStipple can generate BadAlloc, BadGC, BadMatch, and BadPixmap errors.

To set the tile or stipple origin of the specified graphics context, use XSetTSOrigin.

```
XSetTSOrigin(display, gc, ts_x_origin, ts_y_origin)
    Display *display;
    GC gc;
    int ts_x_origin, ts_y_origin;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
ts_x_origin *ts_y_origin*	Specify the x and y coordinates of the tile or stipple origin.

XSetTSOrigin can generate a BadGC error.

5.4.5. Setting the Current Font

To set the current font of the specified graphics context, use XSetFont.

```
XSetFont(display, gc, font)
    Display *display;
    GC gc;
    Font font;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
font	Specifies the font ID.

XSetFont can generate BadAlloc, BadFont, and BadGC errors.

5.4.6. Setting the Clip Region

Xlib provides functions with which you can set the clip origin or the clip mask as well as with which you can set the clip mask to a list of rectangles.

To set the clip origin of the specified graphics context, use XSetClipOrigin.

```
XSetClipOrigin(display, gc, clip_x_origin, clip_y_origin)
    Display *display;
    GC gc;
    int clip_x_origin, clip_y_origin;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
clip_x_origin *clip_y_origin*	Specify the x and y coordinates of the clip origin.

XSetClipOrigin can generate a BadGC error.

To set the clip_mask of the specified graphics context to the specified pixmap, use XSetClip-Mask.

```
XSetClipMask(display, gc, pixmap)
    Display *display;
    GC gc;
    Pixmap pixmap;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
pixmap	Specifies the pixmap.

XSetClipMask can generate BadGC, BadMatch, and BadValue errors.

To set the clip_mask of the specified graphics context to the specified list of rectangles, use XSetClipRectangles.

```
XSetClipRectangles(display, gc, clip_x_origin, clip_y_origin, rectangles, n, ordering)
    Display *display;
    GC gc;
    int clip_x_origin, clip_y_origin;
    XRectangle rectangles[];
    int n;
    int ordering;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
clip_x_origin *clip_y_origin*	Specify the x and y coordinates of the clip origin.
rectangles	Specifies an array of rectangles. These are the rectangles you want to specifies in the graphics context.
n	Specifies the number of rectangles.
ordering	Specifies the ordering relations on the rectangles. Possible values are Unsorted, YSorted, YXSorted, or YXBanded.

The XSetClipRectangles function changes the clip_mask in the specified graphics context to the specified list of rectangles and sets the clip origin. The output is clipped to remain contained within the rectangles. The number of rectangles are specified with the n argument. The clip origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The rectangle coordinates are interpreted relative to the clip origin. The rectangles should be nonintersecting, or the graphics results will be undefined. Note that the list of rectangles can be empty, which effectively disables output. This is the opposite of passing None as the clip_mask in XCreateGC and XChangeGC.

If known by the client, ordering relations on the rectangles can be specified with the ordering argument. This may provide faster operation by the server. If an incorrect ordering is specified, the X server may generate a BadMatch error, but it is not required to do so. If no error is generated, the graphics results are undefined. Unsorted means the rectangles are in arbitrary order. YSorted means that the rectangles are nondecreasing in their Y origin. YXSorted additionally constrains YSorted order in that all rectangles with an equal Y origin are nondecreasing in their X origin. YXBanded additionally constrains YXSorted by requiring that, for every possible Y scan line, all rectangles that include that scan line have identical Y origins and Y extents.

XSetClipRectangles can generate BadAlloc, BadGC, BadMatch, and BadValue errors.

Note

The Xlib library provides a set of basic functions for performing region arithmetic. For information about these functions, see Chapter 10.

5.4.7. Setting the Arc Mode, Subwindow Mode, and Graphics Exposure

To set the arc mode of the specified graphics context, use XSetArcMode.

```
XSetArcMode(display, gc, arc_mode)
    Display *display;
    GC gc;
    int arc_mode;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
arc_mode	Specifies the arc mode: ArcChord specifies that arcs will be chord filled, while ArcPieSlice specifies that arcs will be pie slice filled.

XSetArcMode can generate BadGC and BadValue errors.

To set the subwindow mode of the specified graphics context, use XSetSubwindowMode.

```
XSetSubwindowMode(display, gc, subwindow_mode)
    Display *display;
    GC gc;
    int subwindow_mode;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
subwindow_mode	Specifies the subwindow mode: ClipByChildren clips source and destination by all viewable children, while IncludeInferiors draws through all subwindows, that is, does not clip by inferiors.

XSetSubwindowMode can generate BadGC and BadValue errors.

To set the graphics-exposures flag of the specified graphics context, use XSetGraphicsExposures.

XSetGraphicsExposures(*display*, *gc*, *graphics_exposures*)
    Display **display*;
    GC *gc*;
    Boolean *graphics_exposures*;

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
graphics_exposures	Specifies whether you want GraphicsExpose events to be reported when calling XCopyArea and XCopyPlane with this graphics context. If True, GraphicsExpose events are reported. If False, GraphicsExpose events are not reported.

XSetGraphicsExposures can generate BadGC and BadValue errors.

Chapter 6

Graphics Functions

Once you have connected the display to the X server, you can use the Xlib graphics functions to:

- Clear and copy areas
- Draw points, lines, rectangles, and arcs
- Fill areas
- Manipulate fonts
- Draw text
- Transfer images between clients and server
- Manipulate cursors

If the same drawable and GC is used for each call, Xlib batches back-to-back calls to XDrawPoint, XDrawLine, XDrawRectangle, XFillArc, and XFillRectangle.

6.1. Clearing Areas

Xlib provides functions with which you can clear an area or the entire drawable. Because pixmaps do not have defined backgrounds, they cannot be filled by using the functions described in this Section. Instead, to accomplish an analogous operation on a pixmap, you should use XFillRectangle, which sets the pixmap to a known value. See Section 6.4.1 for information on XFillRectangle.

To clear a specified rectangular area of the specified window, use XClearArea.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.
x *y*	Specify the x and y coordinates. These coordinates are relative to the origin of the window and specifies the upper-left corner of the rectangle.
width *height*	Specify the width and height. These are the dimensions of the rectangle to be cleared.
exposures	Specifies a boolean value of True or False.

The XClearArea function paints a rectangular area in the specified window according to the specified dimensions with the window's background pixel or pixmap. If width is zero, it is replaced with the current width of the window minus x. If height is zero, it is replaced with the current height of the window minus y. If the window has a defined background tile, the rectangle is filled with this tile. If the window has background None, the contents of the window are not changed. In either case, if exposures is True, one or more exposure events are generated for regions of the rectangle that are either visible or are being retained in a backing store. A BadMatch error is generated if you specifies a window whose class is InputOnly.

XClearArea can generate BadMatch, BadValue, and BadWindow errors.

To clear the entire area in the specified window, use XClearWindow.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XClearWindow function clears the entire area in the specified window and is equivalent to XClearArea (display, w, 0, 0, 0, 0, False). If the window has a defined background tile, the rectangle is tiled with a plane-mask of all ones and function of GXcopy. If the window has background None, the contents of the window are not changed. A BadMatch error is generated if you specifies a window whose class is InputOnly.

XClearWindow can generate BadMatch, BadValue, and BadWindow errors.

6.2. Copying Areas

Xlib provides functions with which you can copy an area or a bit-plane.

To copy an area of the specified drawable between drawables of the same root and depth, use XCopyArea.

```
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;
```

display	Specifies the connection to the X server.
src *dest*	Specify the source and destination rectangles to be combined.
gc	Specifies the graphics context.
src_x *src_y*	Specify the x and y coordinates of the source rectangle relative to its origin. These coordinates specify the upper left corner of the source rectangle.
width *height*	Specify the width and height. These are the dimensions of both the source and destination rectangles.
dest_x *dest_y*	Specify the x and y coordinates of the destination rectangle relative to its origin. These coordinates specify the upper left corner of the destination rectangle.

The XCopyArea function combines the specified rectangle of src with the specified rectangle of dest. The rectangles specified by these two arguments must have the same root and depth. Otherwise, XCopyArea generates a BadMatch error. XCopyArea uses these graphics context components: function, plane_mask, subwindow_mode, graphics_exposures, clip_x_origin, clip_y_origin, and clip_mask.

If either the regions of the source rectangle are obscured and have not been retained in backing store or the regions outside the boundaries of the source drawable are specified, those regions are not copied. Instead, the following occurs on all corresponding destination regions that are either visible or are retained in backing store. If the destination rectangle is a window with a background other than None, these corresponding regions of the destination are tiled (with plane_mask of all ones and function GXcopy) with that background. Regardless of tiling or whether the destination is a window or a pixmap, if graphics_exposures is True, then GraphicsExpose events for all corresponding destination regions are generated. If graphics_exposures is True but no regions are exposed, a NoExpose event is generated. Note that by default, graphics_exposures is True on in new GCs. See Chapter 8 for further information.

XCopyArea can generate BadDrawable, BadGC, and BadMatch errors.

To copy a single bit-plane of the specified drawable, use XCopyPlane. The drawable may have different depths.

```
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;
```

display	Specifies the connection to the X server.
src *dest*	Specify the source and destination rectangles to be combined.
gc	Specifies the graphics context.
src_x *src_y*	Specify the x and y coordinates of the source rectangle relative to its origin. These coordinates specify the upper left corner of the source rectangle.
width *height*	Specify the width and height. These are the dimensions of both the source and destination rectangles.
dest_x *dest_y*	Specify the x and y coordinates of the destination rectangle relative to its origin. These coordinates specify the upper left corner of the destination rectangle.
plane	Specifies the bit-plane. You must set exactly one bit.

The XCopyPlane function uses a single bit plane of the specified source rectangle combined with the specified GC to modify the specified rectangle of dest. The rectangles specified by these two arguments must have the same root but need not have the same depth. If the rectangles do not have the same root, a BadMatch error is generated. This function is identical to XCopyArea, except that only one bit-plane is copied. If plane does not have exactly one bit set, a BadValue error is generated.

Effectively, the function forms a pixmap of the same depth as the rectangle of dest and with a size specified by the source region. The function uses the foreground/background pixels in the graphics context (foreground everywhere the bit-plane in src contains a one bit, background everywhere the bit-plane in src contains a zero bit) and the equivalent of a XCopyArea request is performed with all the same exposure semantics. This can also be thought of as using the specified region of the source bit-plane as a stipple with a fill_style of FillOpaqueStippled for filling a rectangular area of the destination.

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

XCopyPlane can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

6.3. Drawing Points, Lines, Rectangles, and Arcs

Xlib provides functions with which you can draw:

- A single point or multiple points
- A single line or multiple lines
- A single rectangle or multiple rectangles
- A single arc or multiple arcs

Some of the functions described in the following subSections use these structures:

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

typedef struct {
        short x, y;
} XPoint;

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

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

All x and y members are signed integers. The width and height members are 16-bit unsigned integers. Applications should be careful not to generate coordinates and sizes out of the 16-bit ranges, as the protocol only has 16 bit fields for these values.

6.3.1. Drawing Single and Multiple Points

To draw a single point in the specified drawable, use XDrawPoint.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates where you want the point drawn.

To draw multiple points in the specified drawable, use XDrawPoints.

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

display	Specifies the connection to the X server.
d	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. CoordModeOrigin treats a coordinates as related to the origin, while CoordModePrevious treats all coordinates after the first as relative to the previous point.

The XDrawPoint and XDrawPoints functions use these graphics context components: function, plane_mask, foreground, subwindow_mode, clip_x_origin, clip_y_origin, and clip_mask. The XDrawPoint function uses the foreground pixel and function components of the graphics context to draw a single point into the specified drawable, while XDrawPoints draws multiple points into the specified drawable. These functions are not affected by the tile or stipple in the graphics context.

When using XDrawPoint, you simply specifies the x and y coordinates where you want the point drawn. For XDrawPoints, you specifies an array of XPoint structures, each of which contains an x and y coordinate. XDrawPoints draws the points in the order listed in the array.

XDrawPoints requires a mode argument that indicates whether the points are relative to the drawable's origin or to the previous point:

- If mode is CoordModeOrigin, all points after the first are relative to the drawable's origin. (The first point is always relative to the drawable's origin.)
- If mode is CoordModePrevious, all points after the first are relative to the previous point.

XDrawPoint can generate BadDrawable, BadGC, and BadMatch errors. XDrawPoint can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

6.3.2. Drawing Single and Multiple Lines

To draw a single line between two points in the specified drawable, use XDrawLine.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x1	
y1	
x2	
y2	Specify the points used to connect the line. Thus, XDrawLine draws a line connecting point x1, y1 to point x2, y2.

To draw multiple lines in the specified drawable, use XDrawLines.

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

display	Specifies the connection to the X server.
d	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. CoordModeOrigin treats a coordinates as related to the origin, while CoordModePrevious treats all coordinates after the first as relative to the previous point.

To draw multiple but not necessarily connected lines in the specified drawable, use XDrawSegments.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
segments	Specifies a pointer to an array of segments.
nsegments	Specifies the number of segments in the array.

The XDrawLine function uses the components of the specified graphics context to draw a line between the specified set of points (x1, y1 and x2, y2). No joining is performed at coincident end points. For any given line, no pixel is drawn more than once. If lines intersect, the intersecting pixels are drawn multiple times.

The XDrawLines function uses the components of the specified graphics context to draw npoints-1 lines between each pair of points (point[i], point[i+1]) in the array of XPoint structures. The lines are drawn in the order listed in the array. The lines join correctly at all intermediate points, and if the first and last points coincide, the first and last lines also join correctly. For any given line, no pixel is drawn more than once. If thin (zero line width) lines intersect, the intersecting pixels will be drawn multiple times. If wide lines intersect, the intersecting pixels are drawn only once, as though the entire PolyLine were a single filled shape. XDrawLines requires a mode argument to determine whether the points are relative to the drawable's origin or to the previous point:

- If mode is CoordModeOrigin, all points after the first are relative to the drawable's origin. (The first point is always relative to the drawable's origin.)
- If mode is CoordModePrevious, all points after the first are relative to the previous point.

The XDrawSegments function draws multiple, but not necessarily connected lines. For each segment, XDrawSegments draws a line between (x1, y1) and (x2, y2). The lines are drawn in the order listed in the array of XSegment structures. No joining is performed at coincident end points. For any given line, no pixel is drawn more than once. If lines intersect, the intersecting pixels are drawn multiple times.

All three functions use 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. The XDrawLines function also uses the join_style graphics context component. All three functions use these graphics context mode-dependent components: foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset, and dash_list. See the general discussion under XCreateGC in Section 5.3.

XDrawLine, XDrawLines, and XDrawSegments can generate BadDrawable, BadGC, and BadMatch errors. XDrawLines can also return BadValue errors.

6.3.3. Drawing Single and Multiple Rectangles

To draw the outline of a single rectangle in the specified drawable, use XDrawRectangle.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates that define the upper left corner of the rectangle.
width *height*	Specify the width and height that define the outline of the rectangle.

To draw the outline of multiple rectangles in the specified drawable, use XDrawRectangles.

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

display	Specifies the connection to the X server.
d	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.

The XDrawRectangle and XDrawRectangles functions draw the outlines of the specified rectangle or rectangles as if a five-point PolyLine were specified for each rectangle:

    [x,y,] [x+width,y] [x+width,y+height] [x,y+height] [x,y]

For the specified rectangle or rectangles, no pixel is drawn more than once. The x and y coordinates of each rectangle are relative to the drawable's origin and define the upper-left corner of the rectangle. XDrawRectangles draws the rectangles in the order listed in the array. If rectangles intersect, the intersecting pixels are drawn multiple times.

XDrawRectangle and XDrawRectangles use these graphics context components: function, plane_mask, line_width, line_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin, and clip_mask. Both functions also use these graphics context mode-dependent components: foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset, and dash_list. See the general discussion under XCreateGC in Chapter 5.

XDrawRectangle and XDrawRectangles can generate BadDrawable, BadGC, and BadMatch errors.

6.3.4. Drawing Single and Multiple Arcs

To draw a single arc in the specified drawable, use XDrawArc.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates. These are the coordinates of the arc and are relative to the origin of the drawable. These coordinates also specifies the upper-left corner of the rectangle.
width *height*	Specify the width and height. These are the major and minor axes of the arc.
angle1	Specifies the start of the arc relative to the three-o-clock position from the center, in units of degrees * 64.
angle2	Specifies the path and extent of the arc relative to the start of the arc, in units of degrees * 64.

To draw multiple arcs in the specified drawable, use XDrawArcs.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
arcs	Specifies a pointer to an array of arcs.
narcs	Specifies the number of arcs in the array.

XDrawArc draws a single circular or elliptical arc, while XDrawArcs draws multiple circular or elliptical arcs. Each 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 degrees scaled by 64 with positive indicating counterclockwise motion and negative indicating clockwise motion. The start of the arc is specified by angle1 relative to the three-o-clock position from the center of the rectangle, and the path and extent of the arc are specified by angle2 relative to the start of the arc. If the magnitude of angle2 is greater than 360 degrees, XDrawArc or XDrawArcs truncates it to 360 degrees.

The x and y coordinates of the rectangle are relative to the origin of the drawable. For an arc specified as [x,y,w,h,a1,a2], the origin of the major and minor axes is at [x+(w/2),y+(h/2)], and the infinitely thin path describing the entire circle or ellipse intersects the horizontal axis at [x,y+(h/2)] and [x+w,y+(h/2)] and intersects the vertical axis at [x+(w/2),y] and [x+(w/2),y+h]. 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 lw, the bounding outlines for filling are given by the infinitely thin paths describing the arcs:

```
[x+dx/2,y+dy/2,w-dx,h-dy,a1,a2]

     and

[x-lw/2,y-lw/2,w+lw,h+lw,a1,a2]

     where

dx=min(lw,w)
dy=min(lw,h)
```

For an arc specified as [x,y,w,h,a1,a2], 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) * w/h) + adjust
```

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

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

In the case of XDrawArc, you simply specifies a single arc. For XDrawArcs, you specifies an array of XArc structures, each of which contains an arc's x and y coordinates, width and height, start position, and path and extent. XDrawArcs draws the arcs in the order listed in the array. For any given arc, no pixel is drawn more than once. If two arcs join correctly and if the line_width is greater than zero and the arcs intersect, no pixel is drawn more than once. Otherwise, the intersecting pixels of intersecting arcs are drawn multiple times. Specifiesing an arc with one endpoint and a clockwise extent draws the same pixels as specifiesing the other endpoint and an equivalent counterclockwise extent, except as it affects joins.

If the last point in one arc coincides with the first point in the following arc, the two arcs will join correctly. If the first point in the first arc coincides with the last point in the last arc, the two arcs will join correctly. By specifiesing one axis to be zero, a horizontal or vertical line can be drawn. Angles are computed based solely on the coordinate system and ignore the aspect ratio.

XDrawArc and XDrawArcs use 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. Both functions also use these graphics context mode-dependent components: foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset, and dash_list.

XDrawArc and XDrawArcs can generate BadDrawable, BadGC, and BadMatch errors.

6.4. Filling Areas

Xlib provides functions with which you can fill:

- A single rectangle or multiple rectangles
- A single polygon
- A single arc or multiple arcs

6.4.1. Filling Single and Multiple Rectangles

To fill a single rectangular area in the specified drawable, use XFillRectangle.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates. These coordinates are relative to the origin of the drawable and specifies the upper-left corner of the rectangle.
width *height*	Specify the width and height. These are the dimensions of the rectangle to be filled.

To fill multiple rectangular areas in the specified drawable, use XFillRectangles.

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

display	Specifies the connection to the X server.
d	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.

The XFillRectangle and XFillRectangles functions fill the specified rectangle or rectangles as if a four-point XFillPolygon were specified for each rectangle:

[x,y] [x+width,y] [x+width,y+height] [x,y+height]

Each function uses the x and y coordinates, width and height dimensions, and graphics context you specifies. Both functions use these graphics context components: function, plane_mask, fill_style, subwindow_mode, clip_x_origin, clip_y_origin, and clip_mask. The two functions also use these graphics context mode_dependent components: foreground, background, tile, stipple, ts_x_origin, and ts_y_origin.

In the case of XFillRectangle, you simply specifies a single rectangle to be filled. For XFillRectangles, you specifies an array of XRectangle structures, each of which contains a rectangle's x and y coordinates and width and height. XFillRectangles fills the rectangles in the order listed in the array. The x and y coordinates of each rectangle are relative to the drawable's origin and define the upper left corner of the rectangle. For any given rectangle, no pixel is drawn more than once. If rectangles intersect, the intersecting pixels are drawn multiple times.

XFillRectangle and XFillRectangles can generate BadDrawable, BadGC, and BadMatch errors.

6.4.2. Filling a Single Polygon

To fill a polygon area in the specified drawable, use XFillPolygon.

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

display	Specifies the connection to the X server.
d	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. You can pass one of these constants: Complex, Convex, or Nonconvex.
mode	Specifies the coordinate mode. CoordModeOrigin treats a coordinates as related to the origin, while CoordModePrevious treats all coordinates after the first as relative to the previous point.

The XFillPolygon function 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 graphics context mode_dependent components: foreground, tile, stipple, ts_x_origin, and ts_y_origin. XFillPolygon fills the region closed by the specified path. The path is closed automatically if the last point in the list does not coincide with the first point. No pixel of the region is drawn more than once.

XFillPolygon requires a mode argument to determine whether the points are relative to the drawable's origin or to the previous point:

- If mode is CoordModeOrigin, all points after the first are relative to the drawable's origin. (The first point is always relative to the drawable's origin.)
- If mode is CoordModePrevious, all points after the first are relative to the previous point.

XFillPolygon also requires a shape argument:

- If shape is Complex, the path may self-intersect.
- If shape is Nonconvex, the path does not self-intersect, but the shape is not wholly convex. If known by the client, specifiesing Nonconvex instead of Complex may improve performance. If you specifies Nonconvex for a self-intersecting path, the graphics results are undefined.
- If shape is Convex, the path is wholly convex. If known by the client, specifiesing Convex can improve performance. If you specifies Convex for a path that is not convex, the graphics results are undefined.

The fill_rule member of the GC controls the filling behavior of self-intersecting polygons. XFillPolygon can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

6.4.3. Filling Single and Multiple Arcs

To fill a single arc in the specified drawable, use XFillArc.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates are relative to the origin of the drawable and specifies the upper-left corner of the rectangle.
width	
height	Specify the width and height. These are the major and minor axes of the arc.
angle1	Specifies the start of the arc relative to the three-o-clock position from the center, in units of degrees * 64.
angle2	Specifies the path and extent of the arc relative to the start of the arc, in units of degrees * 64.

To fill multiple arcs in the specified drawable, use XFillArcs.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
arcs	Specifies a pointer to an array of arcs.
narcs	Specifies the number of arcs in the array.

For each arc, XFillArc or XFillArcs fills the region closed by the infinitely thin path described by the specified arc and, depending on the arc_mode specified in the graphics context one or two line segments. For ArcChord, the single line segment joining the endpoints of the arc is used. For ArcPieSlice, the two line segments joining the endpoints of the arc with the center point are used.

XFillArc and XFillArcs use these graphics context components when filling the arc or arcs: function, plane_mask, fill_style, arc_mode, subwindow_mode, clip_x_origin, clip_y_origin, and clip_mask. The two functions also use these graphics context mode-dependent components: foreground, background, tile, stipple, ts_x_origin, and ts_y_origin.

In the case of XFillArc, you simply specifies a single arc to be filled. For XFillArcs, you specifies an array of XArc structures, each of which contains an arc's x and y coordinates, width and height, start position, and path and extent. XFillArcs fills the arcs in the order listed in the array. For any given arc, no pixel is drawn more than once. If regions intersect, the intersecting pixels are drawn multiple times.

XFillArc and XFillArcs can generate BadDrawable, BadGC, and BadMatch errors.

6.5. Font Metrics

The following Sections discuss how to:

- Load and free fonts
- Obtain and free font names
- Set and return the font search path
- Compute character string sizes
- Return logical extents
- Query character string sizes

The X server loads fonts whenever a program requests a new font. The server can cache fonts for quick lookup. There is never more than one copy of a font stored in the server at one time. Fonts are global across all screens in a server. There are several levels at which one can deal with fonts. Most applications will simply use XLoadQueryFont to load a font and query the font metrics.

Characters in fonts are regarded as masks. Except for image text requests, the only pixels modified are those in which bits are on in the character. This means that it makes sense to draw text using stipples or tiles (for example, many menus gray-out unusable entries).

The XFontStruct structure contains all of the information for the font and consists of the font specific information as well as a pointer to an array of XCharStruct structures for the characters contained in the font. The information in the XFontStruct, XFontProp, and XCharStruct structures is:

```
typedef struct {
        short lbearing;                 /* origin to left edge of raster */
        short rbearing;                 /* origin to right edge of raster */
        short width;                    /* advance to next char's origin */
        short ascent;                   /* baseline to top edge of raster */
        short descent;                  /* baseline to bottom edge of raster */
        unsigned short attributes;      /* per char flags (not predefined) */
} XCharStruct;

typedef struct {
        Atom    name;
        unsigned long card32;
} XFontProp;


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

typedef struct {
        XExtData        *ext_data;           /* hook for extension to hang data */
        Font            fid;                 /* Font id for this font */
        unsigned        direction;           /* hint about the direction font is painted */
        unsigned        min_char_or_byte2;   /* first character */
        unsigned        max_char_or_byte2;   /* last character */
        unsigned        min_byte1;           /* first row that exists */
        unsigned        max_byte1;           /* last row that exists */
        Bool            all_chars_exist;    /* flag if all characters
                                            have nonzero size*/
        unsigned        default_char;       /* char to print for undefined character */
        int             n_properties;       /* how many properties there are */
        XFontProp       *properties;        /* pointer to array of
                                            additional properties*/
        XCharStruct     min_bounds;         /* minimum bounds over all existing char*/
        XCharStruct     max_bounds;         /* maximum bounds over all existing char*/
        XCharStruct     *per_char;          /* first_char to last_char information */
        int             ascent;             /* logical extent above baseline for spacing */
        int             descent;            /* logical decent below baseline for spacing */
} XFontStruct;
```

X supports both single byte/character and two bytes/character text operations. Note that either form can be used with a font, but a single byte/character text requests can only specifies a single byte (that is, the first row of a two byte font). You should view two byte fonts as a two dimensional matrix of defined characters: byte1 specifies the range of defined rows and byte2 defines the range of defined columns of the font. Single byte/character fonts have no rows defined, and the byte2 range specified in the structure defines a range of characters.

The bounding box of a character is defined by the XCharStruct of that character (see below). However, characters may be absent from a font. In this case, the default_char is used. Some fonts may have characters all the same size. In this case, only the information in the XFontStruct characters are used.

The members of the XFontStruct have the following semantics:

- The direction member can be either FontLeftToRight or FontRightToLeft. It is just a hint as to whether most XCharStruct elements have a positive (FontLeftToRight) or a negative (FontRightToLeft) character-width metric. The core protocol defines no support for vertical text.
- If the min_byte1 and max_byte1 members are both zero, min_char_or_byte2 specifies the linear character index corresponding to the first element of the per_char array, and max_char_or_byte2 specifies the linear character index of the last element.

  If either min_byte1 or max_byte1 are nonzero, both min_char_or_byte2 and max_char_or_byte2 will be less than 256, and the 2-byte character index values corresponding to the per_char array element N (counting from 0) are:

      byte1 = N/D + min_byte1
      byte2 = N\D + min_char_or_byte2

  where:

      D = max_char_or_byte2 - min_char_or_byte2 + 1
      / = integer division
      \ = integer modulus

- If the per_char pointer is NULL, all glyphs between the first and last character indexes inclusive have the same information, as given by both min_bounds and max_bounds.
- If all_chars_exist is True, all characters in the per_char array have nonzero bounding boxes.
- The default_char member specifies the character that will be used when an undefined or nonexistent character is used. The default_char is a 16 bit character (not a two byte character). For a font using two byte matrix format, the default_char has byte1 in the most significant byte, and byte2 in the least significant byte. If the default_char itself specifies an undefined or nonexistent character, no printing is performed for an undefined or nonexistent character.

- The min_bounds and max_bounds members contain the most extreme values of each individual XCharStruct component over all elements of this array (ignores nonexistent characters). The bounding box of the font (the smallest rectangle enclosing the shape obtained by superimposing all of the characters at the same origin [x,y]) has its upper-left coordinate at:

  [x + min_bounds.lbearing, y - max_bounds.ascent]

  Its width is:

  max_bounds.rbearing - min_bounds.lbearing

  Its height is:

  max_bounds.ascent + max_bounds.descent

  The x and y are the lower left corner of the box that is relative to the origin.
- The ascent member is the logical extent of the font above the baseline that is used for determining line spacing. Specific characters may extend beyond this.
- The descent member is the logical extent of the font at or below the baseline that is used for determining line spacing. Specific characters may extend beyond this. If the baseline is at Y-coordinate y, the logical extent of the font is inclusive between the Y-coordinate values (y-font.ascent) and (y + font.descent - 1).

For a character origin at [x,y], the bounding box of a character (that is, the smallest rectangle that encloses the character's shape) described in terms of XCharStruct components is a rectangle with its upper-left corner at:

[x + lbearing, y - ascent]

Its width is:

rbearing - lbearing

Its height is:

ascent + descent

The origin for the next character is defined to be:

[x + character-width, y]

Note that the baseline is logically viewed as being just below nondescending characters. When descent is zero, only pixels with Y-coordinates less than y are drawn. And, the origin is logically viewed as being coincident with the left edge of a nonkerned character. When lbearing is zero, no pixels with X-coordinate less than x are drawn). Any of these values could be negative.

The interpretation of the attributes member in the XCharStruct structure is not defined by the core X protocol. A nonexistent character is represented with all members of its XCharStruct set to zero.

A font is not guaranteed to have any properties. The interpretation of the property value (for example, int32, card32) must be derived from a priori knowledge of the property. When possible, fonts should have at least the properties listed in the following table. With atom names, upper/lower case matters. The following builtin property atoms can be found in <X11/Xatom.h>.

Property Name	Type	Description
MIN_SPACE	unsigned	The minimum interword spacing (in pixels).
NORM_SPACE	unsigned	The normal interword spacing (in pixels).
MAX_SPACE	unsigned	The maximum interword spacing (in pixels).

Property Name	Type	Description
END_SPACE	unsigned	The additional spacing (in pixels) at the end of sentences.
SUPERSCRIPT_X SUPERSCRIPT_Y	int	Offset from the character origin where superscripts should begin (in pixels). If the origin is at [x,y], then superscripts should begin at [x + SUPERSCRIPT_X, y - SUPERSCRIPT_Y].
SUBSCRIPT_X SUBSCRIPT_Y	int	Offset from the character origin where subscripts should begin (in pixels). If the origin is at [x,y], then subscripts should begin at [x + SUPERSCRIPT_X, y + SUPERSCRIPT_Y].
UNDERLINE_POSITION	int	Y offset from the baseline to the top of an underline (in pixels). If the baseline is Y-coordinate y, then the top of the underline is at (y + UNDERLINE_POSITION).
UNDERLINE_THICKNESS	unsigned	Thickness of the underline (in pixels)
STRIKEOUT_ASCENT STRIKEOUT_DESCENT	int	Vertical extents for boxing or voiding characters (in pixels). If the baseline is at Y-coordinate y, then the top of the strikeout box is at (y - STRIKEOUT_ASCENT), and the height of the box is (STRIKEOUT_ASCENT + STRIKEOUT_DESCENT).
ITALIC_ANGLE	int	The angle of the dominant staffs of characters in the font, in degrees scaled by 64, relative to the three-o-clock position from the character origin, with positive indicating counterclockwise motion (as in the XDrawArc functions).
X_HEIGHT	int	"1 ex" as in TeX, but expressed in units of pixels. Often the height of lowercase x.
QUAD_WIDTH	int	"1 em" as in TeX, but expressed in units of pixels. Often the width of the digits 0-9.
CAP_HEIGHT	int	Y offset from the baseline to the top of the capital letters, ignoring accents, in pixels. If the baseline is at Y-coordinate y, then the top of the capitals is at (y - CAP_HEIGHT).
WEIGHT	unsigned	The weight or boldness of the font, expressed as a value between 0 and 1000.
POINT_SIZE	unsigned	The point size of this font at the ideal resolution, expressed in 1/10ths of points. There are 72.27 points to the inch.
RESOLUTION	unsigned	The number of pixels per point, expressed in 1/100ths, at which this font was created.

6.5.1. Loading and Freeing Fonts

Xlib provides functions with which you can load fonts, get font information, unload fonts, and free font information. A few font functions use a GContext resource ID or a font ID interchangeably.

To load the specified font, use XLoadFont.

Font XLoadFont(*display*, *name*)
    Display **display*;
    char **name*;

display	Specifies the connection to the X server.
name	Specifies the name of the font. This name is a null terminated string. You should use the ISO Latin-1 encoding, and upper/lower case does not matter.

The XLoadFont function loads the specified font and returns its associated font ID. If the function was unsuccessful at loading the specified font, it generates a BadName error. When the font is no longer needed, the client should call XUnloadFont. Fonts are not associated with a particular screen and can be stored as a component of any graphics context.

XLoadFont can generate BadAlloc and BadName errors.

To obtain the GContext resource ID for the specified GC, use XGContextFromGC.

GContext XGContextFromGC(*gc*)
    GC *gc*;

gc	Specifies the graphics context that you want the resource for.

To return information about a loaded font, use XQueryFont.

XFontStruct *XQueryFont(*display*, *font_ID*)
    Display **display*;
    XID *font_ID*;

display	Specifies the connection to the X server.
font_ID	Specifies the ID of the font or the graphics context.

The XQueryFont function returns a pointer to the XFontStruct structure, which contains information associated with the font. You can either query a font or the font stored in a GC. To free this data, use XFreeFontInfo. In this case, the font ID stored in the XFontStruct will be the ID of the GC, and you need to be careful when using this ID in other functions. For example, the ID of the GC is not valid as a font ID in a call to a Set or Get font function.

To obtain the names and information about loaded fonts, use XListFontsWithInfo.

char **XListFontsWithInfo(*display*, *pattern*, *maxnames*, *count_return*, *info_return*)
    Display **display*;
    char **pattern*;
    int *maxnames*;
    int **count_return*;
    XFontStruct ***info_return*;

display	Specifies the connection to the X server.
pattern	Specifies the null-terminated string associated with the font names that you want returned. You can specify any string, an asterisk "*", or a question mark "?". The asterisk indicates a wildcard on any number of characters, and the question mark indicates a wildcard on a single character.

maxnames	Specifies the maximum number of names that are to be in the returned list.
count_return	Returns the actual number of matched font names.
info_return	Returns a pointer to the font information.

The XListFontsWithInfo function returns a list of names of fonts that match the specified pattern and their associated font information. The list of names is limited to size specified by the max-names argument. The information returned for each font is identical to what XLoadQueryFont would return, except that the per-character metrics are not returned. To free the allocated name array, the client should call XFreeFontNames. To free thefont information array, the client should call XFreeFontInfo.

To free the font information array, use XFreeFontInfo.

```
XFreeFontInfo(names, free_info, actual_count)
    char **names;
    XFontStruct *free_info;
    int actual_count;
```

names	Must be the list of font names returned by XListFontsWithInfo.
free_info	Must be the pointer to font information returned by XListFontWithInfo.
actual_count	Must be the actual number of matched font names returned by XList-FontsWithInfo.

To perform an XLoadFont and XQueryFont in a single operation, use XLoadQueryFont.

```
XFontStruct *XLoadQueryFont(display, name)
    Display *display;
    char *name;
```

display	Specifies the connection to the X server.
name	Specifies the name of the font. This name is a null terminated string.

The XLoadQueryFont function provides the most common way for accessing a font. That is, XLoadQueryFont both opens (loads) the specified font and returns a pointer to the appropriate XFontStruct structure. If the font does not exist, XLoadQueryFont returns NULL.

XLoadQueryFont can generate a BadAlloc error.

To unload the font and free the storage used by the font structure that was allocated by XQueryFont or XLoadQueryFont, use XFreeFont.

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

display	Specifies the connection to the X server.
font_struct	Specifies the storage associated with the font.

The XFreeFont function deletes the association between the font resource ID and the specified font. The font itself will be freed when no other resource references it. The data and the font should not be referenced again.

XFreeFont can generate a BadFont error.

To return the specified font property, use XGetFontProperty.

Bool XGetFontProperty(*font_struct*, *atom*, *value_return*)
    XFontStruct **font_struct*;
    Atom *atom*;
    unsigned long **value_return*;

font_struct	Specifies the storage associated with the font.
atom	Specifies the atom associated with the string name you want returned.
value_return	Returns the value of the font property.

Given the atom for that property, the XGetFontProperty function returns the value of the specified font property. The function returns zero if the atom was not defined or one if it was defined. There are a set of predefined atoms for font properties which can be found in <X11/Xatom.h>. This set contains the standard properties associated with a font. While not guaranteed to be present, it is very likely that the predefined font properties will be present.

To unload the specified font that was loaded by XLoadFont, use XUnloadFont.

XUnloadFont(*display*, *font*)
    Display **display*;
    Font *font*;

display	Specifies the connection to the X server.
font	Specifies the font ID.

The XUnloadFont function deletes the association between the font resource ID and the specified font. The font itself will be freed when no other resource references it. The font may be unloaded on the X server if this is the last reference to the font. In any case, the font should never again be referenced because the resource ID is destroyed.

XUnloadFont can generate a BadFont error.

6.5.2. Obtaining and Freeing Font Names

Xlib provides functions with which you can obtain and free font names. Fonts have font names that usually are related or obvious. These functions let you obtain font names by matching a wild card specification by querying a font type for a list of available sizes, and so on.

To return a list of the available font names, use XListFonts.

char **XListFonts(*display*, *pattern*, *maxnames*, *actual_count_return*)
    Display **display*;
    char **pattern*;
    int *maxnames*;
    int **actual_count_return*;

display	Specifies the connection to the X server.
pattern	Specifies the null-terminated string associated with the font names that you want returned. You can specify any string, an asterisk "*", or a question mark "?". The asterisk indicates a wildcard on any number of characters, and the question mark indicates a wildcard on a single character.
maxnames	Specifies the maximum number of names that are to be in the returned list.
actual_count_return	Returns the actual number of font names.

The XListFonts function returns an array of available font names (as controlled by the font search path; see XSetFontPath) that match the string you passed to the pattern argument. The string should be ISO Latin-1, and upper/lower case does not matter. Each string is terminated by

NULL. The maximum number of names returned in the list depends on the value you passed to the maxnames argument. The function places the actual number of font names returned in the actual_count_return argument. The client should call XFreeFontNames when done with the result to free the memory.

To free a font name array, use XFreeFontNames.

```
XFreeFontNames(list)
    char *list[];
```

list	Specifies the array of strings you want to free.

The XFreeFontNames function frees the array and strings returned by XListFonts.

6.5.3. Setting and Retrieving the Font Search Path

Xlib provides functions with which you can set and retrieve the font search path.

To set the font search path, use XSetFontPath.

```
XSetFontPath(display, directories, ndirs)
    Display *display;
    char **directories;
    int ndirs;
```

display	Specifies the connection to the X server.
directories	Specifies the directory path used to look for a font. Setting the path to the empty list restores the default path defined for the X server.
ndirs	Specifies the number of directories in the path.

The XSetFontPath function defines the directory search path for font lookup. There is only one search path per X server, not one per client. The interpretation of the strings is operating system dependent, but they are intended to specifies directories to be searched in the order listed. Also, the contents of these strings are operating system specific and are not intended to be used by client applications. Usually, the X server is free to cache font information internally rather than having to read fonts from files. As a side-effect of executing this function, the X server is guaranteed to flush all cached information about fonts for which there currently are no explicit resource IDs allocated. The meaning of an error from this request is operating system specific.

XSetFontPath can generate a BadValue error.

To get the current font search path, use XGetFontPath.

```
char **XGetFontPath(display, npaths_return)
    Display *display;
    int *npaths_return;
```

display	Specifies the connection to the X server.
npaths_return	Returns the number of strings in the font path array.

The XGetFontPath function allocates and returns an array of strings containing the search path. The data in the font path should be freed when no longer needed.

To free data returned by XGetFontPath, use XFreeFontPath.

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

list	Specifies the array of strings you want to free.

The XFreeFontPath function, when presented the data from XGetFontPath, frees the data used by the array.

6.5.4. Computing Character String Sizes

Xlib provides functions with which you can compute the width, the logical extents, and the server information about 8-bit and 2-byte text strings. Width is computed by adding the character widths of all of the characters. It does not matter if the font is an 8-bit or 2-byte font.; These functions return the sum of the character metrics, in pixels.

To determine the width of an 8-bit character string, use XTextWidth.

```
int XTextWidth(font_struct, string, count)
    XFontStruct *font_struct;
    char *string;
    int count;
```

font_struct	Specifies the font used for the width computation.
string	Specifies the character string.
count	Specifies the character count in the specified string.

To determine the width of a 2-byte character string, XTextWidth16.

```
int XTextWidth16(font_struct, string, count)
    XFontStruct *font_struct;
    XChar2b *string;
    int count;
```

font_struct	Specifies the font used for the width computation.
string	Specifies the character string.
count	Specifies the character count in the specified string.

6.5.5. Returning Logical Extents

To determine the bounding box of the specified 1-byte character string in the specified font, use XTextExtents.

```
XTextExtents(font_struct, string, nchars, direction_return, font_ascent_return,
        font_descent_return, overall_return)
    XFontStruct *font_struct;
    char *string;
    int nchars;
    int *direction_return;
    int *font_ascent_return, *font_descent_return;
    XCharStruct *overall_return;
```

font_struct	Specifies a pointer to the XFontStruct structure.
string	Specifies the character string.
nchars	Specifies the number of characters in the character string.
direction_return	Returns the value of the direction hint member: FontLeftToRight or FontRightToLeft.
font_ascent_return	Returns the font ascent member.

font_descent_return
Returns the font descent member.

overall_return Returns the overall size in the specified XCharStruct structure.

To return the bounding box of the specified 2-byte character string, use XTextExtents16.

```
XTextExtents16(font_struct, string, nchars, direction_return, font_ascent_return,
        font_descent_return, overall_return)
    XFontStruct *font_struct;
    XChar2b *string;
    int nchars;
    int *direction_return;
    int *font_ascent_return, *font_descent_return;
    XCharStruct *overall_return;
```

font_struct	Specifies a pointer to the XFontStruct structure.
string	Specifies the character string.
nchars	Specifies the number of characters in the character string.
direction_return	Returns the value of the direction hint member: FontLeftToRight or FontRightToLeft.
font_ascent_return	Returns the font ascent member.
font_descent_return	Returns the font descent member.
overall_return	Returns the overall size in the specified XCharStruct structure.

The XTextExtent and XTextExtents16 functions perform the size computation locally and, thereby, avoid the roundtrip overhead of XQueryTextExtents and XQueryTextExtents16. Both functions return an XCharStruct structure, whose members are set to the values as follows.

The ascent member is set to the maximum of the ascent metrics of all characters in the string. The descent member is set to the maximum of the descent metrics. The width member is set to the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string. Let R be the right-side-bearing metric of the character plus W. The lbearing member is set to the minimum L of all characters in the string. The rbearing member is set to the maximum R.

For fonts defined with linear indexing rather than 2-byte matrix indexing, the X server interprets each XChar2b as a 16-bit number that has been transmitted with the most significant byte first. That is, byte1 of the XChar2b is taken as the most significant byte. If the font has no defined default character, undefined characters in the string are taken to have all zero metrics.

6.5.6. Querying Character String Sizes

To query the server for the bounding box of a 1-byte character string, use XQueryTextExtents.

```
XQueryTextExtents(display, font_ID, string, nchars, direction_return, font_ascent_return,
        font_descent_return, overall_return)
    Display *display;
    XID font_ID;
    char *string;
    int nchars;
    int *direction_return;
    int *font_ascent_return, *font_descent_return;
    XCharStruct *overall_return;
```

display	Specifies the connection to the X server.
font_ID	Specifies either the font ID or the graphics context that contains the font.
string	Specifies the character string.
nchars	Specifies the number of characters in the character string.
direction_return	Returns the value of the direction hint member: FontLeftToRight or FontRightToLeft.
font_ascent_return	Returns the font ascent member.
font_descent_return	Returns the font descent member.
overall_return	Returns the overall size in the specified XCharStruct structure.

To query the server for the bounding box of a 2-byte character string in the specified font, use XQueryTextExtents16.

```
XQueryTextExtents16(display, font_ID, string, nchars, direction_return, font_ascent_return,
            font_descent_return, overall_return)
    Display *display;
    XID font_ID;
    XChar2b *string;
    int nchars;
    int *direction_return;
    int *font_ascent_return, *font_descent_return;
    XCharStruct *overall_return;
```

display	Specifies the connection to the X server.
font_ID	Specifies either the font ID or the graphics context that contains the font.
string	Specifies the character string.
nchars	Specifies the number of characters in the character string.
direction_return	Returns the value of the direction hint member: FontLeftToRight or FontRightToLeft.
font_ascent_return	Returns the font ascent member.
font_descent_return	Returns the font descent member.
overall_return	Returns the overall size in the specified XCharStruct structure.

The XQueryTextExtents and XQueryTextExtents16 functions return the bounding box of the specified 8-bit and 16-bit character string in the specified font or the font contained in the specified GC. These functions query the X server and, therefore, suffer the round trip overhead that is avoided by XTextExtents and XTextExtents16. Both functions return a XCharStruct structure whose members are set to the values as follows.

The ascent member is set to the maximum of the ascent metrics of all characters in the string. The descent member is set to the maximum of the descent metrics. The width member is set to the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string. Let L be the left-side-bearing metric of the character plus W. Let R be the right-side-bearing metric of the character plus W. The lbearing member is set to the minimum L of all characters in the string. The rbearing member is set to the maximum R.

Note that the string formal parameter for XQueryTextExtents16 is of type XChar2b, rather than of type char as in XQueryTextExtents. For fonts defined with linear indexing rather than 2-

byte matrix indexing, the X server interprets each XChar2b as a 16-bit number that has been transmitted with the most significant byte first. That is, byte1 of the XChar2b is taken as the most significant byte. If the font has no defined default character, undefined characters in the string are taken to have all zero metrics.

XQueryTextExtents and XQueryTextExtents16 can generate BadFont and BadGC errors.

6.6. Drawing Text

This Section discusses how to draw:

- Complex text
- Text characters
- Image text characters

All of the functions discussed in the following subSections make use of fonts. A font is a graphical description of a set of characters that are used to increase efficiency whenever a set of small, similar sized patterns are repeatedly used.

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

typedef struct {
        XChar2b *chars;                 /* pointer to 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;
```

The fundamental text functions XDrawText and XDrawText16 use these structures. If the font member is None, the font is changed before printing and also is stored in the GC. If an error was generated during text drawing, the font in the GC is undefined.

If you want the baseline (first row of pixels in the descent) to begin at pixel coordinate (x,y), then pass (x,y) as the baseline origin coordinates to the text routines.

For example, consider the background rectangle drawn by XDrawImageString. If you want the upper-left corner of the background rectangle to be at pixel coordinate (x,y), then pass the (x,y+ascent) as the baseline origin coordinates to the text routines. The ascent is the font ascent, as given in the XFontStruct structure. If you want the lower-left corner of the background rectangle to be at pixel coordinate (x,y), then pass the (x,y-descent+1) as the baseline origin coordinates to the text routines. The descent is the font ascent, as given in the XFontStruct structure.

6.6.1. Drawing Complex Text

To draw 8-bit characters in the specified drawable, use XDrawText.

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

display	Specifies the connection to the X server.

d	Specifies the drawable.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the character (initial character origin) and are relative to the origin of the specified drawable.
items	Specifies a pointer to an array of text items.
nitems	Specifies the number of text items in the array.

To draw 2-byte characters in the specified drawable, use XDrawText16.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the character (the initial character origin), and are relative to the origin of the specified drawable.
items	Specifies a pointer to an array of text items.
nitems	Specifies the number of text items in the array.

The XDrawText16 function is like XDrawText, except 2-byte or 16-bit characters are used. These functions allow complex spacing and font shifts between counted string. The XDrawText and XDrawText16 functions use the following elements in the specified GC: function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin, and clip_mask. Both functions also use these graphics context mode-dependent components: foreground, background, tile, stipple, ts_x_origin, and ts_y_origin.

Each text item is processed in turn. A font member other than None in an item causes the font to be stored in the GC and used for subsequent text. A text element delta specifies an additional change in the position along the x axis before the string is drawn. The delta is always added to the character origin, and is not dependent on any characteristics of the font. Each character image, as defined by the font in the GC, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1. If a text item generates a BadFont error, the previous text items may have been drawn.

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 2-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.

XDrawText and XDrawText16 can generate BadDrawable, BadFont, BadGC, and BadMatch errors.

6.6.2. Drawing Text Characters

To draw 8-bit characters in the specified drawable, use XDrawString.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the character (initial character origin) and are relative to the origin of the specified drawable.
string	Specifies the character string.
length	Specifies the number of characters in the string argument.

To draw 2-byte characters in the specified drawable, use XDrawString16.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the character (the initial character origin), and are relative to the origin of the specified drawable.
string	Specifies the character string.
length	Specifies the number of characters in the string argument.

The XDrawString and XDrawString16 functions use these graphics context components: function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin, and clip_mask. Both functions also use these graphics context mode-dependent components: foreground, background, tile, stipple, ts_x_origin, and ts_y_origin. Each character image, as defined by the font in the GC, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1.

XDrawString and XDrawString16 can generate BadDrawable, BadFont, BadGC, and BadMatch errors.

6.6.3. Drawing Image Text Characters

Some applications, in particular terminal emulators, need to print image text in which both the foreground and background bits of each character are painted. This avoids annoying flicker on many displays.

To draw 8-bit image text characters in the specified drawable, use XDrawImageString.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the image text character (initial character origin) and are relative to the origin of the specified drawable.
string	Specifies the character string.
length	Specifies the number of characters in the string argument.

To draw 2-byte image text characters in the specified drawable, XDrawImageString16.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the image text character and are relative to the origin of the specified drawable.
string	Specifies the character string.
length	Specifies the number of characters in the string argument.

The XDrawImageText16 function is like XDrawImageText, except 2-byte or 16-bit characters are used. These functions also modify both the foreground and background pixels in the characters.

XDrawImageString and XDrawImageString16 use these graphics context components: plane_mask, foreground, background, font, subwindow_mode, clip_x_origin, clip_y_origin, and clip_mask. The effect is first to fill a destination rectangle with the background pixel defined in the GC and then to paint the text with the foreground pixel. The upper-left corner of the filled rectangle is at:

[x,y - font_ascent]

The width is:

overall_width

The height is:

font_ascent + font_descent

The overall_width, font_ascent, and font_descent are as would be returned by XQueryTextExtents using gc and string. The function and fill_style defined in the GC are ignored for these functions. The effective function is GXcopy, and the effective fill_style is FillSolid.

Note that the string formal parameter for XDrawImageString16 is of type XChar2b, rather than of type char as in XDrawImageString. For fonts defined with 2-byte matrix indexing and used with XDrawImageString, each byte is used as a byte2 with a byte1 of zero.

XDrawImageString and XDrawImageString16 can generate BadDrawable, BadGC, and BadMatch errors.

6.7. Transferring Images Between Client and Server

Xlib provides functions with which you can transfer images between a client and the server. Because the server may require diverse data formats, Xlib provides an image object that fully describes the data in memory and that provides for basic operations on that data. You should reference the data through the image object, not the data directly. However, some implementations of the Xlib library may efficiently deal with frequently used data formats by replacing routines in the procedure vector with special case routines. Supported operations include destroying the image, getting a pixel, storing a pixel, extracting a sub image of an image, and adding a constant to an image. See Chapter 10 for further information about these utility functions. Chapter 10 also has information about additional library functions used to perform basic operations on an image.

All the image manipulation functions discussed in this Section make use of the XImage data structure. The members in the XImage structure are:

```
typedef struct _XImage {
        int width, height;              /* size of image */
        int xoffset;                    /* number of pixels offset in X direction */
        int format;                     /* XYBitmap, XYPixmap, ZPixmap */
        char *data;                     /* pointer to image data */
        int byte_order;                 /* data byte order, LSBFirst, MSBFirst */
        int bitmap_unit;                /* quant. of scanline 8, 16, 32 */
        int bitmap_bit_order;           /* LSBFirst, MSBFirst */
        int bitmap_pad;                 /* 8, 16, 32 either XY or ZPixmap */
        int depth;                      /* depth of image */
        int bytes_per_line;             /* accelerator to next line */
        int bits_per_pixel;             /* bits per pixel (ZPixmap) */
        unsigned long red_mask;         /* bits in z arrangement */
        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;

The XImage structure describes an image as it exists in the client's memory. The user may request that some of the members (for example, height, width, and xoffset) be changed when the image is sent to the server. That is, the user may send a subset of the image. Other members (for example, byte_order, bitmap_unit, and so forth) are characteristics of both the image and of the server. If these members differ between the image and the server, XPutImage makes the appropriate conversions. If the image is formatted as an XYPixmap, (that is, the format member is set to XYPixmap), the first byte of the first line of plane n is located at the address (data + (n * height * bytes_per_line)).

To combine an image in memory with a rectangle of a drawable on your display, use XPutImage.

```
XPutImage(display, d, gc, image, src_x, src_y, dst_x, dst_y, width, height)
      Display *display;
      Drawable d;
      GC gc;
      XImage *image;
      int src_x, src_y;
      int dst_x, dst_y;
      unsigned int width, height;
```

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
image	Specifies the image you want combined with the rectangle.
src_x	Specifies the offset in X from the left edge of the image defined by the XImage data structure.
src_y	Specifies the offset in from the top edge of the image defined by the XImage data structure.
dst_x *dst_y*	Specify the x and y coordinates. These are the coordinates of the subimage and are relative to the origin of the drawable where the image will be drawn.
width *height*	Specify the width and height. These are of the subimage and define the dimensions of the rectangle.

The XPutImage function combines an image in memory with a rectangle of the specified drawable. The function uses these graphics context components: function, plane_mask, subwindow_mode, clip_x_origin, clip_y_origin, and clip_mask. This function also uses the graphics context mode-dependent components foreground and background. If XYBitmap format is used, the depth must be one, and the image must be XYFormat. Otherwise, XPutImage generates a BadMatch error. The foreground pixel in the GC defines the source for one bits in the image, and the background pixel defines the source for the zero bits. For XYPixmap and ZPixmap, the depth must match the depth of drawable. Otherwise, it generates a BadMatch error. For XYPixmap, the image must be sent in XYFormat. For ZPixmap, the image must be sent in the ZFormat defined for the given depth. The Section of the image defined by the src_x, src_y, width, and height arguments are drawn on the specified part of the drawable.

XPutImage can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

To return the contents of a rectangle in the specified drawable on the display, use XGetImage. This function is intended specifically for rudimentary screen dump support.

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
x *y*	Specify the x and y coordinates. These coordinates define the upper left corner of the rectangle and are relative to the origin of the drawable.
width *height*	Specify the width and height of the subimage. These arguments define the dimensions of the rectangle.
plane_mask	Specifies the plane mask.
format	Specifies the format for the image. You can pass one of these constants: XYPixmap or ZPixmap.

The XGetImage function returns the XImage structure. This structure provides you with the contents of the specified rectangle of the drawable in the format you specify. If the format argument is XYPixmap, the function transmits only the bit planes you passed to the plane_mask argument. If the plane_mask argument only requests a subset of the planes of the display, the depth of the returned image will be the number of planes requested. If the format argument is ZPixmap, the function transmits as zero the bits in all planes not specified in the plane_mask argument. The function performs no range checking on the values in plane_mask and ignores extraneous bits.

XGetImage returns the depth of the image to the depth member of the XImage structure. The depth of the image is as specified when the drawable was created.

If the drawable is a pixmap, then the given rectangle must be wholly contained within the pixmap, otherwise the X server will generate a BadMatch error. If the drawable is a window, the window must be mapped, and if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen and wholly contained within the outside edges of the window. Otherwise, XGetImage generates a BadMatch error. Note that the borders of the window can be included and read with this request. 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.

XGetImage can generate BadDrawable, BadMatch, and BadValue errors.

To copy the contents of a rectangle on the display to the specified location within a pre-existing image structure, use XGetSubImage.

```
XImage *XGetSubImage(display, d, x, y, width, height, plane_mask, format, dest_image, dest_x,
            dest_y)
    Display *display;
    Drawable d;
    int x, y;
    unsigned int width, height;
    unsigned long plane_mask;
    int format;
    XImage *dest_image;
    int dest_x, dest_y;
```

display	Specifies the connection to the X server.
d	Specifies the drawable.
x	
y	Specify the x and y coordinates. These coordinates define the upper left corner of the rectangle and are relative to the origin of the drawable.
width	
height	Specify the width and height of the subimage. These arguments define the dimensions of the rectangle.
plane_mask	Specifies the plane mask.
format	Specifies the format for the image. You can pass one of these constants: XYPixmap or ZPixmap.
dest_image	Specify the the destination image.
dest_x	
dest_y	Specify the x and y coordinates of the destination rectangle relative to its origin. These coordinates specify the upper left corner of the destination rectangle. These coordinates determine where the subimage will be placed within the destination image.

The XGetSubImage function updates the Image with the specified subimage in the same manner as XGetImage. If the format argument is XYPixmap, the function transmits only the bit planes you passed to the plane_mask argument. If the format argument is ZPixmap, the function transmits as zero the bits in all planes not specified in the plane_mask argument. The function performs no range checking on the values in plane_mask and ignores extraneous bits.

XGetSubImage does not update any fields in the destination XImage structure. The depth of the destination XImage structure must be the same as that of the drawable. Otherwise, XGetSubImage generates a BadMatch error. 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, 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, it generates a BadMatch error. If the window has a backing-store, then 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.

XGetSubImage can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

6.8. Manipulating Cursors

Xlib provides functions with which you can manipulate the cursor. This Section discusses how to:

- Create a cursor
- Change or destroy a cursor

- Define the cursor for a window

These functions allow you to load and change cursors associated with windows. Each window can have a different cursor defined for it. Whenever the pointer is in a visible window, it will be set to the cursor defined for that window. If no cursor was defined for that window, the cursor will be that defined for the parent window.

From X's perspective, a cursor consists of a cursor shape, mask, colors for the shape and mask, and "hot spot". The cursor pixmap determines the shape of the cursor and must be a depth of one. The mask pixmap determines the bits which will be modified by the cursor. The colors determine the colors of the shape and mask. The hot spot defines the point on the cursor which will be reported when a pointer event occurs. There may be and probably are limitations imposed by the hardware on cursors as to size and whether a mask is implemented. XQueryBestCursor can be used to find out what sizes are possible. In the future, it is intended that most standard cursors will be stored as a special font. Client programs refer to cursors by using Cursor resource IDs.

6.8.1. Creating a Cursor

Xlib provides functions with which you can create a font, bitmap, or glyph cursor. XRecolorCursor can be used to change the color of the cursor to the desired colors. To create a cursor from two bitmaps, use XCreatePixmapCursor. This is a fundamental request used by XCreateFontCursor.

To create a cursor from a standard font, use XCreateFontCursor.

```
#include <X11/cursorfont.h>

Cursor XCreateFontCursor(display, shape)
    Display *display;
    unsigned int shape;
```

display	Specifies the connection to the X server.
shape	Specifies the shape in which you want to create the standard cursor.

X provides a set of standard cursor shapes in a special font named cursorfont. Applications are encouraged to use this interface for their cursors because the font can be customized for the individual display type. The shape argument specifies which glyph of the standard fonts to use.

The hotspot comes from the information stored in the cursor font. The initial colors of a cursor are a black foreground and a white background. See Appendix B for further information about cursors and their shapes in fonts.

XCreateFontCursor can generate BadAlloc, BadMatch, and BadValue errors.

To create a cursor from font glyphs, use XCreateGlyphCursor.

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

display	Specifies the connection to the X server.
source_font	Specifies the font for the source glyph.
mask_font	Specifies the font for the mask glyph. You can also pass None.
source_char	Specifies the character glyph for the source.

mask_char Specifies the glyph character for the mask.

foreground_color
Specifies the red, green, and blue (RGB) values for the foreground of the source.

background_color
Specifies the red, green, and blue (RGB) values for the background of the source.

The XCreateGlyphCursor function is similar to XCreatePixmapCursor and creates a cursor from font glyphs. For XCreateGlyphCursor, however, the source and mask bitmaps are obtained from the specified font glyphs. The source_char must be a defined glyph in source_font, and, if mask_font is given, mask_char must be a defined glyph in mask_font. Otherwise, XCreateGlyphCursor generates a BadValue error. The mask_font and character are optional. The origins of the source_char and mask_char (if defined) glyphs are positioned coincidently and define the hotspot. The source_char and mask_char need not have the same bounding box metrics, and there is no restriction on the placement of the hotspot relative to the bounding boxes. If no mask_char is given, all pixels of the source are displayed. You can free the fonts immediately by calling XFreeFont if no further explicit references to them are to be made.

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

XCreateGlyphCursor can generate BadAlloc, BadFont, and BadValue errors.

6.8.2. Changing and Destroying Cursors

Xlib provides functions with which you can change the cursor color, destroy the cursor, and determine the cursor size.

To change the color of the specified cursor, use XRecolorCursor.

```
XRecolorCursor(display, cursor, foreground_color, background_color)
    Display *display;
    Cursor cursor;
    XColor *foreground_color, *background_color;
```

display Specifies the connection to the X server.

cursor Specifies the cursor.

foreground_color
Specifies the red, green, and blue (RGB) values for the foreground of the source.

background_color
Specifies the red, green, and blue (RGB) values for the background of the source.

The XRecolorCursor function changes the color of the specified cursor, and, if the cursor is being displayed on a screen, the change is visible immediately.

XRecolorCursor can generate a BadCursor error.

To free (destroy) the specified cursor, use XFreeCursor.

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

display Specifies the connection to the X server.

cursor Specifies the cursor.

The XFreeCursor function deletes the association between the cursor resource ID and the specified cursor. The cursor storage is freed when no other resource references it. The specified cursor ID should not be referred to again or an error will be generated.

XFreeCursor can generate a BadCursor error.

To determine useful cursor sizes, use XQueryBestCursor.

```
Status XQueryBestCursor(display, d, width, height, width_return, height_return)
    Display *display;
    Drawable d;
    unsigned int width, height;
    unsigned int *width_return, *height_return;
```

display	Specifies the connection to the X server.
d	Specifies the drawable. The drawable indicates the desired screen.
width *height*	Specify the width and height of the cursor that you want the size information for.
width_return *height_return*	Returns the best width and height (that is, closest to the specified width and height).

Some displays allow larger cursors than other displays. XQueryBestCursor provides a way to find out what size cursors are actually possible on the display. This function returns the largest size that can be displayed. For a cursor shape, it returns a bitmap shape acceptable for XCreatePixmapCursor. Applications should be prepared to use smaller cursors on displays which cannot support large ones.

XQueryBestCursor can generate a BadDrawable error.

6.8.3. Defining the Cursor

Xlib provides functions with which you can define or undefine which cursor is to be displayed in a window.

To define which cursor will be used in a window, use XDefineCursor.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.
cursor	Specifies the cursor that is to be displayed when the pointer is in the specified window. You can pass None if no cursor is to be displayed.

If a cursor is set, it will be used when the pointer is in the window.

XDefineCursor can generate BadAlloc, BadCursor, and BadWindow errors.

To undefine the mouse cursor in the specified window, use XUndefineCursor.

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

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XUndefineCursor undoes the effect of a previous XDefineCursor for this window. When the mouse is in the window, the parent's cursor will now be used. On the root window, with no cursor specified, the default cursor is restored.

XUndefineCursor can generate a BadWindow error.

Chapter 7

Window Manager Functions

Once you have connected the display to the X server and want to create a window manager application, you can use the Xlib window manager functions to:

- Change the parent of a window
- Control the lifetime of a window
- Determine resident colormaps
- Grab the pointer
- Grab the keyboard
- Grab the server
- Control event processing
- Manipulate the keyboard encoding
- Manipulate the screen saver
- Control host access

Note

The functions in this Chapter are most often used by window managers, though it can be difficult to definitively categorize functions as application only or window manager only. It is not expected that these functions will be used by most normal application programs.

7.1. Changing the Parent of a Window

To change a window's parent within a single screen, use XReparentWindow. There is no way to move a window between screens.

```
XReparentWindow(display, w, parent, x, y)
    Display *display;
    Window w;
    Window parent;
    int x, y;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
parent	Specifies the parent window ID.
x	
y	Specify the x and y coordinates of the position in the new parent window of the specified window.

The XReparentWindow function reparents the specified window by inserting it as the child of the specified parent. If the window is mapped, XReparentWindow automatically performs an XUnmapWindow request on the specified window. The function then removes the specified window from its current position in the hierarchy and inserts it as the child of the specified parent. The window is placed on top in the stacking order with respect to sibling windows.

After reparenting the specified window, XReparentWindow causes the X server to generate a ReparentNotify event. The override_redirect member of the structure returned by this event is set to either True or False. Window manager clients normally should ignore this event if this member is set to True. See Chapter 8 for more information on ReparentNotify event processing. Finally, if the specified window was originally mapped, this function automatically performs an XMapWindow request on it.

The X server performs normal exposure processing on formerly obscured windows. The X server might not generate exposure events for regions from the initial XUnmapWindow request that are immediately obscured by the final XMapWindow request. A BadMatch error is generated:

- If the new parent window is not on the same screen as the old parent window;
- If the new parent window is the specified window or an inferior of the specified window; or
- If the specified window has a ParentRelative background and if the new parent window is not the same depth as the specified window.

XReparentWindow can generate BadMatch and BadWindow errors.

7.2. Controlling the Lifetime of a Window

The save-set of a client is a list of other client's windows which, if they are inferiors of one of the client's windows at connection close, should not be destroyed and should be remapped if they are unmapped. To allow an application's window to survive when a window manager that has reparented a window fails, Xlib provides the save-set functions with which you can change a client's save-set, add a subwindow to a client's save-set, or remove a subwindow from a client's save-set.

The functions described in this Section are used to control the longevity of subwindows that are normally destroyed when the parent is destroyed. For example, a window manager that wants to add decoration to a window by adding a frame might reparent an application's window. When the frame is destroyed, the application's window should not be destroyed, but returned to its previous place in the window hierarchy.

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

To add or remove a window from the client's save-set, use XChangeSaveSet.

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

display	Specifies the connection to the X server.
w_add	Specifies the window ID of the window whose children you want to add to the client's save-set.
change_mode	Specifies the mode. You can pass one of SetModeInsert or SetModeDelete. If SetModeInsert, XChangeSaveSet adds the window to this client's save-set. If SetModeDelete, XChangeSaveSet deletes the window from this client's save-set.

Depending on the constant you passed to the change_mode argument, the XChangeSaveSet function either adds or removes a subwindow from the client's save-set. The specified window must have been created by some other client. Otherwise, XChangeSaveSet generates a BadMatch error. See Section 2.6 for information on what happens to the save-set during connection close. The X server automatically removes windows from the save-set when they are destroyed.

XChangeSaveSet can generate BadMatch, BadValue, and BadWindow errors.

To add a window to the client's save-set, use XAddToSaveSet.

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

display	Specifies the connection to the X server.
w_add	Specifies the window ID of the window whose children you want to add to the client's save-set.

The XAddToSaveSet function adds the children of the specified window to the client's save-set. The specified window must have been created by some other client. Otherwise, XAddToSaveSet generates a BadMatch error. See Section 2.6 for information on what happens to the save-set during connection close. The X server automatically removes windows from the save-set when they are destroyed.

XAddToSaveSet can generate BadMatch and BadWindow errors.

To remove a window from the client's save-set, use XRemoveFromSaveSet.

```
XRemoveFromSaveSet(display, w_remove)
    Display *display;
    Window w_remove;
```

display	Specifies the connection to the X server.
w_remove	Specifies the window ID of the window whose children you want to remove from the client's save-set.

The XRemoveFromSaveSet function removes the children of the specified window from the client's save-set. The specified window must have been created by some other client. Otherwise, XRemoveFromSaveSet generates a BadMatch error. See Section 2.6 for information on what happens to the save-set during connection close. The X server automatically removes windows from the save-set when they are destroyed.

XRemoveFromSaveSet can generate BadMatch and BadWindow errors.

7.3. Determining Resident Colormaps

Xlib provides functions with which you can install a colormap, uninstall a colormap, and obtain a list of installed colormaps.

Window manager applications usually install and uninstall colormaps. These tasks should not be performed by normal client applications. The X server always maintains a subset of the installed colormaps in an ordered list called the "required list." The length of the required list is at most the minimum installed maps specified for the screen when the connection is opened to the server. The X server maintains the required list as follows:

- If you pass a colormap resource ID to the cmap argument of XInstallColormap, it adds the colormap to the top of the list and truncates a colormap at the bottom of the list, if necessary, so as not to exceed the maximum length of the list.
- If you pass a colormap resource ID to the cmap argument of XInstallColormap and that colormap is in the required list, the colormap is removed from the required list. A colormap is not added to the required list when it is installed implicitly by the server, and the X server cannot implicitly uninstall a colormap that is in the required list.

Initially, only the default colormap for a screen is installed, but it is not in the required list.

To install a colormap, use XInstallColormap.

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

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.

The XInstallColormap function installs the specified color map for its associated screen. All windows associated with this colormap immediately display with true colors. You associated the windows with this colormap when you created them by calling the functions XCreateWindow or XCreateSimpleWindow. See Chapter 3 for a discussion of how to associate a window with a colormap using these functions.

If the specified colormap is not already an installed colormap, the X server generates a ColormapNotify event on every window having the cmap as the colormap resource ID. In addition, for every other colormap that is installed or uninstalled as a result of a call to this function, the X server generates a ColormapNotify event on every window having that cmap as the colormap resource ID.

XInstallColormap can generate a BadColor error.

To uninstall a colormap, use XUninstallColormap.

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

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.

The XUninstallColormap function removes the specified color map from the required list for its screen. As a result, the specified cmap might be uninstalled, and the X server might implicitly install or uninstall additional colormaps. Which colormaps get installed or uninstalled is sever-dependent, except that the required list must remain installed.

If the specified colormap becomes uninstalled, the X server generates a ColormapNotify event on every window having the cmap as the colormap resource ID. In addition, for every other colormap that is uninstalled as a result of a call to this function, the X server generates a ColormapNotify event on every window having that cmap as the colormap resource ID.

XUninstallColormap can generate a BadColor error.

To obtain a list of the currently installed colormaps, use XListInstalledColormaps.

```
Colormap *XListInstalledColormaps(display, w, num_return)
    Display *display;
    Window w;
    int *num_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose screen you want to obtain the list of currently installed colormaps.
num_return	Returns the list of currently installed color maps.

The XListInstalledColormaps function returns a list of the currently installed color maps for the screen of the specified window. The order in which the colormaps appear in the list is not significant, and there is no explicit indication of the required list. You should free the allocated list by using XFree, when it is no longer needed. (See Section 2.4 for further information.)

XListInstalledColormaps can generate a BadWindow error.

7.4. Grabbing the Pointer

Xlib provides functions with which you can control input from the pointer, which usually is a mouse. Window managers most often use these facilities to implement certain styles of user interfaces. Some toolkits also need to use these facilities for special purposes. However, most applications will not use these functions.

Usually, the X server delivers keyboard and mouse events as soon as they occur to the appropriate client depending upon the window and input focus. The X server provides sufficient control over event delivery to allow window managers to support mouse ahead and various other styles of user interface. Many of these depend upon synchronous delivery of events. The delivery of pointer and keyboard events can be controlled independently.

When mouse buttons or keyboard keys are grabbed, events will be sent to the grabbing client rather than the normal client who would have received the event. If the keyboard or pointer is in asynchronous mode, further mouse and keyboard events will continue to be processed. If the keyboard or pointer is in synchronous mode, no further events will be processed until the grabbing client allows them. (See XAllowEvents.) The keyboard or pointer is considered frozen during this interval. The triggering event can also be replayed.

Note that the logical state of a device (as seen by means of the X protocol) may lag the physical state if device event processing is frozen.

There are two kinds of grabs: active and passive. An active grab occurs when a single client grabs the keyboard and/or pointer explicitly. (See XGrabPointer and XGrabKeyboard.) Clients can also grab a particular keyboard key or pointer button in a window. The grab activates when the key or button is actually pressed, and is called a ''passive grab''. Passive grabs can be very convenient for implementing reliable pop-up menus. For example, you can arrange to guarantee that the pop-up will be mapped before the up pointer button event occurs by grabbing a button requesting synchronous behavior. The down event will trigger the grab and freeze further processing of pointer events until you have the chance to map the pop up window. You can then allow further event processing. The up event will then be correctly processed relative to the pop-up window.

For many operations, there are functions which take a time argument. The X server includes a timestamp in various events. One special time called CurrentTime represents the current server time. The X server maintains the time when the input focus was last changed and the time of the server itself when the client last performed an active grab (discussed below), when the keyboard was last grabbed, when the pointer was last grabbed, or when a selection was last changed. Your application may be slow reacting to an event. You often need some way to specifies that your request not occur if some other application has in the meanwhile taken control of the keyboard, pointer, or selection. By providing the timestamp from the event in the request, you can arrange that the operation not take effect if someone else has in the meanwhile performed an operation.

To grab the pointer, use XGrabPointer.

```
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;
```

display	Specifies the connection to the X server.

grab_window	Specifies the window ID of the window relative to which events are reported while it is grabbed.
owner_events	Specifies if the pointer events are to be reported normally (pass True) or with respect to the grab window if selected by the event mask (pass False).
event_mask	Specifies which pointer events are reported to the client. They can be the bitwise inclusive OR of these pointer event mask bits: ButtonPressMask, ButtonReleaseMask, EnterWindowMask, LeaveWindowMask, PointerMotionMask, PointerMotionHintMask, Button1MotionMask, Button2MotionMask, Button3MotionMask, Button4MotionMask, Button5MotionMask, ButtonMotionMask, KeyMapStateMask.
pointer_mode	Controls further processing of pointer events. You can pass one of these constants: GrabModeSync or GrabModeAsync.
keyboard_mode	Controls further processing of keyboard events. You can pass one of these constants: GrabModeSync or GrabModeAsync.
confine_to	Specifies the window to confine the pointer in or None if it is not to be confined.
cursor	Specifies the cursor that is to be displayed during the grab.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or CurrentTime.

The XGrabPointer function actively grabs control of the pointer and returns GrabSuccess if the grab was successful. Further pointer events are only reported to the grabbing client. This function overrides any active pointer grab by this client. If owner_events is False, all generated pointer events are reported with respect to grab_window and are only reported if selected by event_mask. If owner_events is True, and if a generated pointer event would normally be reported to this client, it is reported normally. Otherwise, the event is reported with respect to the grab_window and is only reported if selected by event_mask. For either value of owner_events, unreported events are discarded.

Pointer_mode controls further processing of pointer events, and keyboard_mode controls further processing of main keyboard events. If the pointer_mode is GrabModeAsync, pointer event processing continues normally. If the pointer is currently frozen by this client, the processing of events for the pointer is resumed. If the pointer_mode is GrabModeSync, the state of the pointer, as seen by client applications, appears to freeze, and no further pointer events are generated by the X server until the grabbing client calls the XAllowEvents function or until the pointer grab is released. Actual pointer changes are not lost while the pointer is frozen; they are simply queued for later processing.

If the keyboard_mode is GrabModeAsync, keyboard event processing is unaffected by activation of the grab. If the keyboard_mode is GrabModeSync, then the state of the keyboard, as seen by client applications, appears to freeze, and no further keyboard events are generated by the X server until the grabbing client calls the XAllowEvents function or until the pointer grab is released. Actual keyboard changes are not lost while the pointer is frozen; they are simply queued for later processing.

If a cursor is specified, it is displayed regardless of what window the pointer is in. If no cursor is specified, the normal cursor for that window is displayed when the pointer is in grab_window or one of its subwindows; otherwise, the cursor for grab_window is displayed.

If a confine_to window is specified, the pointer will be restricted to stay contained in that window. The confine_to window need have no relationship to the grab_window. If the pointer is not initially in the confine_to window, it is warped automatically to the closest edge just before the grab activates and enter/leave events are generated normally. 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 allows you to 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 specifies the timestamp from the event, the second application may successfully grab the pointer, while the first gets an indication the other application grabbed the pointer before its request was processed.

XGrabPointer generates EnterNotify and LeaveNotify events.

The XGrabPointer function fails and returns:

GrabNotViewable	If grab_window or confine_to window is not viewable.
AlreadyGrabbed	If the pointer is actively grabbed by some other client.
GrabFrozen	If the pointer is frozen by an active grab of another client.
GrabInvalidTime	If the specified time is earlier than the last-pointer-grab time or later than the current X server time. Otherwise, the last-pointer-grab time is set to the specified time and CurrentTime is replaced by the current X server time.

XGrabPointer can generate BadCursor, BadValue, and BadWindow errors.

To grab a mouse button, use XGrabButton.

XGrabButton(*display*, *button_grab*, *modifiers*, *grab_window*, *owner_events*, *event_mask*,
        pointer_mode, *keyboard_mode*, *confine_to*, *cursor*)
    Display **display*;
    unsigned int *button_grab*;
    unsigned int *modifiers*;
    Window *grab_window*;
    Bool *owner_events*;
    unsigned int *event_mask*;
    int *pointer_mode*, *keyboard_mode*;
    Window *confine_to*;
    Cursor *cursor*;

display	Specifies the connection to the X server.
button_grab	Specifies the pointer button that is to be grabbed when the specified modifier keys are down. You can pass AnyButton, which is equivalent to issuing the grab request for all possible buttons.
modifiers	Specifies the set of keymasks. This mask is the bitwise inclusive OR of these keymask bits: ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask. You can also pass AnyModifier, which is equivalent to issuing the grab request for all possible modifier combinations (including the combination of no modifiers).
grab_window	Specifies the window ID of the window you want to grab.
owner_events	Specifies if the pointer events are to be reported normally (pass True) or with respect to the grab window if selected by the event mask (pass False).
event_mask	Specifies which pointer events are reported to the client. They can be the bitwise inclusive OR of these pointer event mask bits: ButtonPressMask, ButtonReleaseMask, EnterWindowMask, LeaveWindowMask, PointerMotionMask, PointerMotionHintMask, Button1MotionMask, Button2MotionMask, Button3MotionMask, Button4MotionMask, Button5MotionMask, ButtonMotionMask, KeyMapStateMask.
pointer_mode	Controls further processing of pointer events. You can pass one of these constants: GrabModeSync or GrabModeAsync.

keyboard_mode Controls further processing of keyboard events. You can pass one of these constants: GrabModeSync or GrabModeAsync.

confine_to Specifies the window to confine the pointer in or None if it is not to be confined.

cursor Specifies the cursor that is to be displayed during the grab.

The XGrabButton function establishes a passive grab. In the future:

- IF the pointer is not grabbed and the specified button is logically pressed when the specified modifier keys logically are down (and no other buttons or modifier keys logically are down)
- AND if grab_window contains the pointer
- AND if the confine_to window (if any) is viewable
- AND if a passive grab on the same button/key combination does not exist on any ancestor of the grab window
- THEN the pointer is actively grabbed, as for XGrabPointer, the last-pointer-grab time is set to the time at which the button was pressed (as transmitted in the ButtonPress event), and the ButtonPress events is reported.

The interpretation of the remaining arguments is as for XGrabPointer. The active grab is terminated automatically when the logical state of the pointer has all button released (independent of the state of the logical modifier keys).

Note that the logical state of a device (as seen by means of the X protocol) may lag the physical state if device event processing is frozen.

A modifiers of AnyModifier is equivalent to issuing the grab request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned keycodes. A button of AnyButton is equivalent to issuing the request for all possible buttons. Otherwise, it is not required that the specified button currently be assigned to a physical button.

The request fails and the X server generates a BadAccess error 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 generating an BadAccess error (no grabs are established) if there is a conflicting grab for any combination. The request has no effect on an active grab.

XGrabButton can generate BadAlloc, BadCursor, BadValue, and BadWindow errors.

To ungrab a mouse button, use XUngrabButton.

XUngrabButton(*display*, *button_ungrab*, *modifiers*, *ungrab_window*)
  Display **display*;
  unsigned int *button_ungrab*;
  unsigned int *modifiers*;
  Window *ungrab_window*;

display Specifies the connection to the X server.

button_ungrab Specifies the pointer button that is to be released in combination with the modifier keys. You can pass AnyButton, which is equivalent to issuing the ungrab request for all possible buttons.

modifiers Specifies the set of keymasks. This mask is the bitwise inclusive OR of these keymask bits: ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask. You can also pass AnyModifier, which is equivalent to issuing the ungrab request for all possible modifier combinations (including the combination of no modifiers).

ungrab_window Specifies the window ID of the window you want to ungrab.

The XUngrabButton function releases the passive button/key combination on the specified window if it was grabbed by this client. A modifier of AnyModifier is equivalent to issuing the ungrab request for all possible modifier combinations, including the combination of no modifiers. A button of AnyButton is equivalent to issuing the request for all possible buttons. This function has no effect on an active grab.

XUngrabButton can generate a BadWindow error.

To ungrab the pointer, use XUngrabPointer.

```
XUngrabPointer(display, time)
    Display *display;
    Time time;
```

display	Specifies the connection to the X server.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or CurrentTime.

The XUngrabPointer function releases the pointer and any queued events, if this client has actively grabbed the pointer from XGrabPointer, XGrabButton, or from a normal button press. The function does not release the pointer if the specified time is earlier than the last-pointer-grab time or is later than the current X server time. It also generates EnterNotify and LeaveNotify events. The X server performs an XUngrabPointer automatically if the event window or confine-to window for an active pointer grab becomes not viewable.

To change the active pointer grab, use XChangeActivePointerGrab.

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

display	Specifies the connection to the X server.
event_mask	Specifies which pointer events are reported to the client. They can be the bitwise inclusive OR of these pointer event mask bits: ButtonPressMask, ButtonReleaseMask, EnterWindowMask, LeaveWindowMask, PointerMotionMask, PointerMotionHintMask, Button1MotionMask, Button2MotionMask, Button3MotionMask, Button4MotionMask, Button5MotionMask, ButtonMotionMask, KeyMapStateMask.
cursor	Specifies the cursor. This is the cursor that is displayed. A possible value you can pass is None.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or CurrentTime.

The XChangeActivePointerGrab function changes the specified dynamic parameters if the pointer is actively grabbed by the client and if the specified time is no earlier than the last-pointer-grab time and no later than the current X server time. This function has no effect on the passive parameters of an XGrabButton. The event-mask is always augmented to include ButtonPress mask and ButtonRelease mask. The interpretation of event_mask and cursor is the same as described in XGrabPointer.

XChangeActivePointerGrab can generate a BadCursor error.

7.5. Grabbing the Keyboard

Xlib provides functions with which you can grab or ungrab the keyboard as well as allow events.

To grab the keyboard, use XGrabKeyboard.

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*;

display	Specifies the connection to the X server.
grab_window	Specifies the window ID of the window associated with the keyboard you want to grab.
owner_events	Specifies a boolean value of either True or False.
pointer_mode	Controls further processing of pointer events. You can pass one of these constants: GrabModeSync or GrabModeAsync.
keyboard_mode	Controls further processing of keyboard events. You can pass one of these constants: GrabModeSync or GrabModeAsync.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or CurrentTime.

The XGrabKeyboard function actively grabs control of the main keyboard and generates FocusIn and FocusOut events. Further key events are reported only to the grabbing client. This function overrides any active keyboard grab by this client. If owner_events is False, all generated key events are reported with respect to grab_window. If owner_events is True, then if a generated key event would normally be reported to this client, it is reported normally; otherwise the event is reported with respect to the grab_window. Both KeyPress and KeyRelease events are always reported, independent of any event selection made by the client.

The pointer_mode argument controls the further processing of pointer events, and the keyboard_mode argument controls the further processing of the keyboard events.

- If the keyboard_mode argument is GrabModeAsync, keyboard event processing continues normally; if the keyboard is currently frozen by this client, then processing of keyboard events is resumed. If the keyboard_mode argument is GrabModeSync, the state of the keyboard (as seen by client applications) appears to freeze, and no further keyboard events are generated by the server until the grabbing client issues a releasing XAllowEvents call or until the keyboard grab is released.. Actual keyboard changes are not lost while the keyboard is frozen; they are simply queued for later processing.
- If pointer_mode is GrabModeAsync, pointer event processing is unaffected by activation of the grab. If pointer_mode is GrabModeSync, the state of the pointer (as seen by client applications) appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing XAllowEvents call or until the keyboard grab is released. Actual pointer changes are not lost while the pointer is frozen; they are simply queued for later processing.

XGrabKeyboard fails and returns:

AlreadyGrabbed	If the keyboard is actively grabbed by some other client.
GrabNotViewable	If grab_window is not viewable.
GrabInvalidTime	If the specified time is earlier than the last-keyboard-grab time or later than the current X server time. Otherwise, the last-keyboard-grab time is set to the specified time and CurrentTime is replaced by the current X server time.
GrabFrozen	If the keyboard is frozen by an active grab of another client.

XGrabKeyboard can generate BadValue and BadWindow errors.

To ungrab the keyboard, use XUngrabKeyboard.

```
XUngrabKeyboard(display, time)
    Display *display;
    Time time;
```

display	Specifies the connection to the X server.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or CurrentTime.

The XUngrabKeyboard function releases the keyboard and any queued events if this client has it actively grabbed from either XGrabKeyboard or XGrabKey. The function does not release the keyboard and any queued events if the specified time is earlier than the last-keyboard-grab time or is later than the current X server time. It also generates FocusIn and FocusOut events. The X server automatically performs an XUngrabKeyboard if the event window for an active keyboard grab becomes not viewable.

To passively grab a single key of the keyboard key, use XGrabKey.

```
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;
```

display	Specifies the connection to the X server.
keycode	Specifies the keycode. This keycode maps to the specific key you want to grab. You can pass either the keycode or AnyKey.
modifiers	Specifies the set of keymasks. This mask is the bitwise inclusive OR of these keymask bits: ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask. You can also pass AnyModifier, which is equivalent to issuing the grab key request for all possible modifier combinations (including the combination of no modifiers).
grab_window	Specifies the window ID of the window associated with the keys you want to grab.
owner_events	Specifies a boolean value of either True or False.
pointer_mode	Controls further processing of pointer events. You can pass one of these constants: GrabModeSync or GrabModeAsync.
keyboard_mode	Controls further processing of keyboard events. You can pass one of these constants: GrabModeSync or GrabModeAsync.

The XGrabKey function establishes a passive grab on the keyboard. Consequently, in the future:

- IF the keyboard is not grabbed and the specified key, which itself can be a modifier key, is logically pressed when the specified modifier keys logically are down (and no other keys are down),
- AND no other modifier keys logically are down,

- AND EITHER the grab window is an ancestor of (or is) the focus window OR the grab window is a descendent of the focus window and contains the pointer,
- AND a passive grab on the same key combination does not exist on any ancestor of the grab window,
- THEN the keyboard is actively grabbed, as for XGrabKeyboard, the last-keyboard-grab time is set to the time at which the key was pressed (as transmitted in the KeyPress event), and the KeyPress event is reported.

The interpretation of the remaining arguments is as for XGrabKeyboard. The active grab is terminated automatically when logical state of the keyboard has the specified key released (independent of the logical state of the modifier keys).

Note that the logical state of a device (as seen by means of the X protocol) may lag the physical state if device event processing is frozen.

A modifier of AnyModifier is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned keycodes. A key of AnyKey is equivalent to issuing the request for all possible keycodes. Otherwise, the key must be in the range specified by min_keycode and max_keycode in the connection setup. Otherwise, XGrabKey generates a BadValue error.

A BadAccess error is generated if some other client has issued an XGrabKey with the same key combination on the same window. When using AnyModifier or AnyKey, the request fails completely and the X server generates a BadAccess error and no grabs are established if there is a conflicting grab for any combination.

XGrabKey can generate BadAccess, BadValue, and BadWindow errors.

To ungrab a key, use XUngrabKey.

```
XUngrabKey(display, keycode, modifiers, ungrab_window)
    Display *display;
    int keycode;
    unsigned int modifiers;
    Window ungrab_window;
```

display	Specifies the connection to the X server.
keycode	Specifies the keycode. This keycode maps to the specific key you want to ungrab. You can pass either the keycode or AnyKey.
modifiers	Specifies the set of keymasks. This mask is the bitwise inclusive OR of these keymask bits: ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask. You can also pass AnyModifier, which is equivalent to issuing the ungrab key request for all possible modifier combinations (including the combination of no modifiers).
ungrab_window	Specifies the window ID of the window associated with the keys you want to ungrab.

The XUngrabKey function releases the key combination on the specified window if it was grabbed by this client. This function has no effect on an active grab. A modifier of AnyModifier is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). A key of AnyKey is equivalent to issuing the request for all possible nonmodifier key codes.

XUngrabKey can generate a BadWindow error.

To allow further events to be processed when the device has been frozen, use XAllowEvents.

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

display	Specifies the connection to the X server.
event_mode	Specifies the event mode. You can pass one of these constants: AsyncPointer, SyncPointer, AsyncKeyboard, SyncKeyboard, ReplayPointer, ReplayKeyboard, AsyncBoth, or SyncBoth.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or CurrentTime.

The XAllowEvents function releases some queued events if the client has caused a device to freeze. The function has no effect if the specified time is earlier than the last-grab time of the most recent active grab for the client, or if the specified time is later than the current X server time. The following describes the processing that occurs depending on what constant you pass to the event_mode argument:

AsyncPointer	If the pointer is frozen by the client the pointer event processing continues as usual. If the pointer is frozen twice by the client on behalf of two separate grabs, AsyncPointer thaws for both. AsyncPointer has no effect if the pointer is not frozen by the client, but the pointer need not be grabbed by the client.
SyncPointer	If the pointer is frozen and actively grabbed by the client, pointer event processing continues normally 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, the pointer does not freeze. SyncPointer has no effect if the pointer is not frozen by the client or if the pointer is not grabbed by the client.
ReplayPointer	If the pointer is actively grabbed by the client and is frozen as a result of an event having been sent to the client (either from the activation of an XGrabButton or from a previous XAllowEvents with mode (SyncPointer, but not from an XGrabPointer), the pointer grab is released and that event is completely reprocessed. This time, however, the function ignores any passive grabs at or above (towards the root) the grab_window of the grab just released. The request has no effect if the pointer is not grabbed by the client or if the pointer is not frozen as the result of an event.
AsyncKeyboard	If the keyboard is frozen by the client, the keyboard event processing continues as usual. If the keyboard is frozen twice by the client on behalf of two separate grabs, AsyncKeyboard "thaws" for both. AsyncKeyboard has no effect if the keyboard is not frozen by the client, but the keyboard need not be grabbed by the client.
SyncKeyboard	If the keyboard is frozen and actively grabbed by the client, keyboard event processing continues as usual 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, the keyboard does not freeze. SyncKeyboard has no effect if the keyboard is not frozen by the client or if the keyboard is not grabbed by the client.

ReplayKeyboard	If the keyboard is actively grabbed by the client and is frozen as the result of an event having been sent to the client (either from the activation of a XGrabKey or from a previous XAllowEvents with mode (SyncKeyboard, but not from a XGrabKeyboard), the keyboard grab is released and that event is completely reprocessed. This time, however, the function ignores any passive grabs at or above (towards the root) the grab_window of the grab just released. The request has no effect if the keyboard is not grabbed by the client or if the keyboard is not frozen as the result of an event.
SyncBoth	If both pointer and keyboard are frozen by the client, event processing (for both devices) continues normally until the next ButtonPress, ButtonRelease, KeyPress, or KeyRelease event is reported to the client for a grabbed device (button event for the pointer, key event for the keyboard), at which time the devices again appear to freeze. However, if the reported event causes the grab to be released, then the devices do not freeze (but if the other device is still grabbed, then a subsequent event for it will still cause both devices to freeze). 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	If the pointer and the keyboard are frozen by the client, event processing (for both devices) continues normally. If a device is frozen twice by the client on behalf of two separate grabs, AsyncBoth "thaws" for both. AsyncBoth has no effect unless both pointer and keyboard are frozen by the client.

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 possible for a single device to be frozen because of both grabs. In this case, the freeze must be released on behalf of both grabs before events can again be processed.

XAllowEvents can generate a BadValue error.

7.6. Grabbing the Server

Xlib provides functions with which you can grab and ungrab the server. These functions can be used to control processing of output on other connections by the window system server. No processing of requests or close downs on any other connection will occur while the server is grabbed. A client closing its connection automatically ungrabs the server. Although grabbing the server is highly discouraged, it is sometimes necessary.

To grab the server, use XGrabServer.

XGrabServer(*display*)
    Display **display*;

display Specifies the connection to the X server.

The XGrabServer function disables processing of requests and close-downs on all other connections than the one this request arrived on. It is recommended that you not grab the X server any more than is absolutely necessary.

To ungrab the server, use XUngrabServer.

```
XUngrabServer(display)
    Display *display;
```

display	Specifies the connection to the X server.

The XUngrabServer function restarts processing of requests and close-downs on other connections. You should avoid grabbing the server as much as possible.

7.7. Miscellaneous Control Functions

This Section discusses how to:

- Control the input focus
- Control the pointer
- Kill clients

7.7.1. Controlling the Input Focus

Xlib provides functions with which you can move the pointer position as well as set and get the input focus.

To move the pointer to an arbitrary point on the screen, use XWarpPointer.

```
XWarpPointer(display, src_w, dest_w, src_x, src_y, src_width, src_height, dest_x,
        dest_y)
     Display *display;
     Window src_w, dest_w;
     int src_x, src_y;
     unsigned int src_width, src_height;
     int dest_x, dest_y;
```

display	Specifies the connection to the X server.
src_w	Specifies the window ID of the source window. You can pass the window ID or None.
dest_w	Specifies the window ID of the destination window. You can pass the window ID or None.
src_x *src_y*	Specify the x and y coordinates within the source window.
src_width *src_height*	Specify the width and height of the source window.
dest_x *dest_y*	Specify the x and y coordinates within the destination window.

The XWarpPointer function moves the pointer to the coordinates specified by the dest_x and dest_y arguments, relative to the destination window's origin. If the destination window is None, the pointer is moved by offsets specified by the dest_x and dest_y coordinates. There is seldom any reason for calling this function. The pointer should normally be left to the mouse. If you do use this function, however, it generates events just as if the user had instantaneously moved the pointer from one position to another. Note that you cannot use XWarpPointer to move the pointer outside the confine_to window of an active pointer grab. An attempt to do so will only move the pointer as far as the closest edge of the confine_to window.

If the src_w argument is None, the move is independent of the current pointer position. However, if src_w is a window, the move only takes place if the pointer is currently contained in a visible portion of the specified rectangle of the src_w. If dest_w is None, the function moves the pointer by offsets (dest_x, dest_y) relative to the current position of the pointer. If dest_w is a window, the function moves the pointer to (dest_x, dest_y) relative to the origin of dest_w.

However, if src_w is not None, the move only takes place if src_w contains the pointer and the pointer is currently contained in the specified rectangle of the src_w.

The coordinates passed to src_x and src_y are relative to the source window's origin. If src_height is zero, the function replaces it with the current height of the source window minus src_y. If src_width is zero, the function replaces it with the current width of the source window minus src_x.

XWarpPointer can generate a BadWindow error.

To set the input focus, use XSetInputFocus.

XSetInputFocus(*display*, *focus*, *revert_to*, *time*)
    Display **display*;
    Window *focus*;
    int *revert_to*;
    Time *time*;

display	Specifies the connection to the X server.
focus	Specifies the window ID. This is the window in which you want to set the input focus. You can pass the window ID or either PointerRoot or None.
revert_to	Specifies which window the input focus reverts to if the window becomes not viewable. You can pass one of these constants: RevertToParent, RevertToPointerRoot, or RevertToNone.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or CurrentTime.

The XSetInputFocus function changes the input focus and the last-focus-change time. The function has no effect if the specified time is earlier than the current last-focus-change time or is later than the current X server time. Otherwise, the last-focus-change time is set to the specified time and the CurrentTime replaced by the current X server time. This function causes the X server to generate FocusIn and FocusOut events.

Depending on what value you assign to the focus argument, XSetInputFocus executes as follows:

- If you assign None to the focus argument, all keyboard events are discarded until you set a new focus window. In this case, the revert_to argument is ignored.
- If you assign a window ID to the focus argument, it becomes the keyboard's focus window. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported normally. Otherwise, the event is reported relative to the focus window.
- If you assign PointerRoot to the focus argument, the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event. In this case, the revert_to argument is ignored.

The specified focus window must be viewable at the time XSetInputFocus is called. Otherwise, it generates a BadMatch error. If the focus window later becomes not viewable, the X server evaluates the revert_to argument to determine the new focus window:

- If you assign RevertToParent to the revert_to argument, the focus reverts to the parent (or the closest viewable ancestor), and the new revert_to value is taken to be RevertToNone.
- If you assign RevertToPointerRoot or RevertToNone to the revert_to argument, the focus reverts to that value. When the focus reverts, the X server generates FocusIn and FocusOut events, but the last-focus-change time is not affected.

XSetInputFocus can generate BadMatch, BadValue, and BadWindow errors.

To obtain the current input focus, use XGetInputFocus.

```
XGetInputFocus(display, focus_return, revert_to_return)
    Display *display;
    Window *focus_return;
    int *revert_to_return;
```

display	Specifies the connection to the X server.
focus_return	Returns the focus window ID, or either PointerRoot, or None.
revert_to_return	Returns the current focus state. The function can return one of these constants: RevertToParent, RevertToPointerRoot, or RevertToNone.

The XGetInputFocus function returns the focus window ID and the current focus state.

7.7.2. Controlling the Pointer

Xlib provides functions with which you can change the pointer control or can get the current pointer control parameters.

To control the interactive feel of the pointer device, use XChangePointerControl.

```
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;
```

display	Specifies the connection to the X server.
do_accel	Specifies a boolean value that controls whether the values for the accel_numerator or accel_denominator are set. You can pass one of these constants: True or False.
do_threshold	Specifies a boolean value that controls whether the value for the accel_numerator or accel_denominator are set. You can pass one of these constants: True or False.
accel_numerator	Specifies the numerator for the acceleration multiplier.
accel_denominator	Specifies the denominator for the acceleration multiplier.
threshold	Specifies the acceleration threshold.

The XChangePointerControl function defines how the pointing device moves. The acceleration, expressed as a fraction, is a multiplier for movement. For example, specifiesing 3/1 means the pointer moves three times as fast as normal. The fraction may be rounded arbitrarily by the X server. Acceleration only takes effect if the pointer moves more than threshold pixels at once and only applies to the amount beyond the value in the threshold argument. Setting a value to -1 restores the default. The values of the do_accel and do_threshold arguments must be nonzero for the pointer values to be set. Otherwise, the parameters will be unchanged. Negative values generate a BadValue error, as does a zero value for the accel_denominator argument.

XChangePointerControl can generate a BadValue error.

To get the current pointer parameters, use XGetPointerControl.

XGetPointerControl(*display*, *accel_numerator_return*, *accel_denominator_return*,
        threshold_return)
    Display **display*;
    int **accel_numerator_return*, **accel_denominator_return*;
    int **threshold_return*;

display Specifies the connection to the X server.

accel_numerator_return
Returns the numerator for the acceleration multiplier.

accel_denominator_return
Returns the denominator for the acceleration multiplier.

*threshold_return*Returns the acceleration threshold.

The XGetPointerControl function returns the pointer's current acceleration multiplier and acceleration threshold.

7.7.3. Killing Clients

Xlib provides functions with which you can control the life time of resources owned by a client or can cause the connection to a client to be destroyed.

To change the close down mode of a client, use XSetCloseDownMode.

XSetCloseDownMode(*display*, *close_mode*)
    Display **display*;
    int *close_mode*;

display Specifies the connection to the X server.

close_mode Specifies the client close down mode you want to change. You can pass one of these constants: DestroyAll, RetainPermanent, or RetainTemporary.

The XSetCloseDownMode defines what will happen to the client's resources at connection close. A connection starts in DestroyAll mode. See the Section 2.6 for information on what happens to the client's resources when the close_mode argument is one of the valid constants.

XSetCloseDownMode can generate a BadValue error.

To destroy a client, use XKillClient.

XKillClient(*display*, *resource*)
    Display **display*;
    XID *resource*;

display Specifies the connection to the X server.

resource Specifies any resource associated with the client you want to destroy. You can also pass AllTemporary.

The XKillClient function forces a close-down of the client that created the resource if a valid resource is specified. If the client has already terminated in either RetainPermanent or Retain-Temporary mode, all of the client's resources are destroyed. If AllTemporary is specified, the resources of all clients that have terminated in RetainTemporary are destroyed. See Section 2.6 for further information.

XKillClient can generate a BadValue error.

7.8. Keyboard Settings

Xlib provides functions with which you can change the keyboard control, obtain a list of the auto-repeat keys, turn keyboard auto-repeat on or off, ring the bell, set or obtain the pointer button or keyboard mapping, and obtain a bit vector for the keyboard.

This Section discusses the user preference options of bell, keyclick, mouse behavior, and so on. The default values for many of these functions are determined by command line arguments to the X server, and on UNIX-based systems are typically set in the /etc/ttys file. Not all implementations will actually be able to control all of these parameters.

The XChangeKeyboardControl function operates on a XKeyboardControl structure:

```
/* 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)

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

The following describes each of the members of the XKeyboardControl structure.

The key_click_percent member sets the volume for key clicks between 0 (off) and 100 (loud) inclusive, if possible. A setting of -1 restores the default. Other negative values generate a BadValue error.

The bell_percent sets the base volume for the bell between 0 (off) and 100 (loud) inclusive, if possible. A setting of -1 restores the default. Other negative values generate a BadValue error. The bell_pitch member sets the pitch (specified in Hz) of the bell, if possible. A setting of -1 restores the default. Other negative values generate a BadValue error. The bell_duration member sets the duration, specified in milliseconds, of the bell, if possible. A setting of -1 restores the default. Other negative values generate a BadValue error.

If both the led_mode and led members are specified, the state of those LEDs are changed, if possible. If only led_mode is specified, the state of all LEDs are changed, if possible. At most 32 LEDs numbered from one are supported. No standard interpretation of LEDs is defined. A BadMatch error is generated if an led is specified without an led_mode.

If both the auto_repeat_mode and key members are specified, the auto_repeat_mode of that key is changed, if possible (where LED is the ordinal number of the LED to be changed, not a mask). If only auto_repeat_mode is specified, the global auto_repeat mode for the entire keyboard is changed, if possible, and does not affect the per_key settings. A BadMatch error is generated if a key is specified without an auto_repeat_mode.

A bell generator connected with the console but not directly on a keyboard is treated as if it were part of the main keyboard. The order in which controls are verified and altered is server-dependent. If an error is generated, a subset of the controls may have been altered.

To change control from a keyboard, use XChangeKeyboardControl.

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

display	Specifies the connection to the X server.
value_mask	Specifies one value for each one bit in the mask (least to most significant bit). The values are associated with the set of keys for the previously specified keyboard.
values	Specifies a pointer to the structure XKeyboardControl.

The XChangeKeyboardControl function controls the keyboard characteristics defined by the XKeyboardControl structure. The values argument specifies which values are to be changed. The value_mask contains one value for each one bit in the mask (least to most significant bit).

XChangeKeyboardControl can generate BadMatch and BadValue errors.

To obtain the current control values for the keyboard, use XGetKeyboardControl.

```
XGetKeyboardControl(display, values_return)
    Display *display;
    XKeyboardState *values_return;
```

display	Specifies the connection to the X server.
values_return	Returns the current keyboard parameter in the specified XKeyboardState structure.

The XGetKeyboardControl function returns the current control values for the keyboard to the XKeyboardState structure. The members of this structure are:

```
typedef struct {
        int key_click_percent;
        int bell_percent;
        unsigned int bell_pitch, bell_duration;
        unsigned long led_mask;
        int global_auto_repeat;
        char auto_repeats[32];
} XKeyboardState;
```

For the LEDs, the least significant bit of led_mask corresponds to LED one, and each one bit in led_mask indicates an LED that is lit. The auto_repeats member is a bit vector. Each one bit 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. The global_auto_repeat member can be set to either AutoRepeatModeOn or AutoRepeatModeOff.

To turn on keyboard auto-repeat, use XAutoRepeatOn.

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

display	Specifies the connection to the X server.

The XAutoRepeatOn function turns on auto-repeat for the keyboard on the specified display.

To turn off keyboard auto-repeat, use XAutoRepeatOff.

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

display Specifies the connection to the X server.

The XAutoRepeatOff function turns off auto-repeat for the keyboard on the specified display.

To ring the bell, use XBell.

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

display Specifies the connection to the X server.

percent Specifies the base volume for the bell, which can range from -100 to 100 inclusive.

The XBell function rings the bell on the keyboard on the specified display, if possible. The specified volume is relative to the base volume for the keyboard. If the value for the percent argument is not in the range -100 to 100 inclusive, a BadValue error is generated. The volume at which the bell is rung when the percent argument is nonnegative is:

base - [(base * percent) / 100] + percent

The volume at which the bell is rung when the percent argument is negative is:

base + [(base * percent) / 100]

To change the base volume of the bell for this keyboard, use XChangeKeyboardControl.

XBell can generate a BadValue error.

To set the mapping of buttons on the pointer, use XSetPointerMapping.

```
int XSetPointerMapping(display, map, nmap)
    Display *display;
    unsigned char map[];
    int nmap;
```

display Specifies the connection to the X server.

map Specifies the mapping list.

nmap Specifies the number of items in mapping list.

The XSetPointerMapping function sets the mapping of the pointer and causes the X server to generate a MappingNotify event on a status of MappingSuccess. Elements of the list are indexed starting from one. The length of the list must be the same as XGetPointerMapping would return. Otherwise, XSetPointerMapping generates a BadValue error. The index is a core button number, and the element of the list defines the effective number. A zero element disables a button, and elements are not restricted in value by the number of physical buttons. However, no two elements can have the same nonzero value. Otherwise, it generates a BadValue error. If any of the buttons to be altered are logically in the down state, the status reply is MappingBusy and the mapping is not changed. This function returns either MappingSuccess or MappingBusy.

XSetPointerMapping can generate a BadValue error.

To get the pointer mapping, use XGetPointerMapping.

```
int XGetPointerMapping(display, map, nmap)
    Display *display;
    unsigned char map[];
    int nmap;
```

display Specifies the connection to the X server.

map	Specifies the mapping list.
nmap	Specifies the number of items in mapping list.

The XGetPointerMapping function returns the current mapping of the pointer. Elements of the list are indexed starting from one. The length of the list indicates the number of physical buttons. The nominal mapping for a pointer is the identity mapping: map[i]=i.

To obtain a bit vector that describes the state of the keyboard, use XQueryKeymap.

```
XQueryKeymap(display, keys_return)
    Display *display;
    char keys_return[32];
```

display	Specifies the connection to the X server.
keys_return	Returns an array of bytes that identifies which keys are pressed down. Each bit represents one key of the keyboard.

The XQueryKeymap function returns a bit vector for the logical state of the keyboard, where each one bit indicates that the corresponding key is currently pressed down. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N+7 with the least significant bit in the byte representing key 8N.

Note that the logical state of a device (as seen by means of the X protocol) may lag the physical state if device event processing is frozen.

7.9. Keyboard Encoding

Most applications will find the simple interface, XLookupString which will perform simple translation of a key event to an ASCII string most useful. Keyboard related utilities are discussed in Chapter 10. The following Section explains how to completely control the bindings of symbols to keys and modifiers.

A KeyCode represents a physical (or logical) key. Keycodes lie in the inclusive range [8,255]. A keycode value carries no intrinsic information, although server implementors may attempt to encode geometry (for example, matrix) information in some fashion so that it can be interpreted in a server-dependent fashion. The mapping between keys and keycodes cannot be changed.

A KeySym is an encoding of a symbol on the cap of a key. The set of defined KeySyms include the ISO Latin character sets (1-4), Katakana, Arabic, Cyrillic, Greek, Technical, Special, Publishing, APL, Hebrew, and a special miscellany of keys found on keyboards (RETURN, HELP, TAB, and so on). To the extent possible, these sets are derived from international standards. In areas where no standards exist, some of these sets are derived from Digital standards. The list of defined symbols can be found in the <X11/keysymdef.h> header file. Unfortunately, some C preprocessors have limits on the number of defined symbols. If you must use keysyms not in the Latin 1-4, Greek, and miscellany classes, you may have to define a symbol for those sets. Most applications usually only include <X11/keysym.h>, which defines symbols for ISO Latin 1-4, Greek, and Miscellany.

A list of KeySyms is associated with each KeyCode. The length of the list can vary with each KeyCode. The list is intended to convey the set of symbols on the corresponding key. By convention, if the list contains a single KeySym and if that KeySym is alphabetic and case distinction is relevant for it, then it should be treated as equivalent to a two-element list of the lowercase and uppercase KeySyms. For example, if the list contains the single KeySym for uppercase A, the client should treat it as if it were a pair with lowercase "a" as the first KeySym and uppercase "A" as the second KeySym.

For any KeyCode, the first KeySym in the list should be chosen as the interpretation of a KeyPress when no modifier keys are down. The second KeySym in the list normally should be chosen when the Shift modifier is on, or when the Lock modifier is on and Lock is interpreted as ShiftLock. When the Lock modifier is on and is interpreted as CapsLock, it is suggested that the

Shift modifier first be applied to choose a KeySym, but if that KeySym is lowercase alphabetic, the corresponding uppercase KeySym should be used instead. Other interpretations of CapsLock are possible; for example, it may be viewed as equivalent to ShiftLock, but only applying when the first KeySym is lowercase alphabetic and the second KeySym is the corresponding uppercase alphabetic. No interpretation of KeySyms beyond the first two in a list is suggested here. No spatial geometry of the symbols on the key is defined by their order in the KeySym list, although a geometry might be defined on a vendor-specific basis. The X server does not use the mapping between KeyCodes and KeySyms. Rather, it stores it merely for reading and writing by clients.

To obtain the symbols for the specified keycodes, use XGetKeyboardMapping.

KeySym *XGetKeyboardMapping(*display*, *first_keycode_wanted*, *keycode_count*,
keysyms_per_keycode_return)
Display **display*;
KeyCode *first_keycode_wanted*;
int *keycode_count*;
int **keysyms_per_keycode_return*;

display Specifies the connection to the X server.

first_keycode_wanted
Specifies the first keycode that is to be returned.

keycode_count Specifies the number of keycodes that are to be returned.

keysyms_per_keycode_return
Returns the number of keysyms per keycode.

The XGetKeyboardMapping function, starting with first_keycode, returns the symbols for the specified number of keycodes. The value specified in the first_keycode argument must be greater than or equal to min_keycode as returned in the Display structure at connection setup. Otherwise, XGetKeyboardMapping generates a BadValue error. In addition, the following expression must be less than or equal to max_keycode as returned in the Display structure at connection setup. If this is not the case, a BadValue error is generated.

first_keycode + keycode_count - 1

The number of elements in the keysyms list is:

keycode_count * keysyms_per_keycode_return

Then, KeySym number N, counting from zero, for keycode K has an index, counting from zero, of the following in KeySym:

(K - first_code) * keysyms_per_code + N

The keysyms_per_keycode_return value is chosen arbitrarily by the X 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.

To free the storage returned by XGetKeyboardMapping, use XFree. (See Section 2.4 for further information.)

XGetKeyboardMapping can generate a BadValue error.

To change the keyboard mapping, use XChangeKeyboardMapping.

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

display	Specifies the connection to the X server.
first_keycode	Specifies the first keycode that is to be changed.
keysyms_per_keycode	Specifies the keysyms that are to be used.
keysyms	Specifies a pointer to an array of keysyms.
num_codes	Specifies the number of keycodes that are to be changed.

The XChangeKeyboardMapping function, starting with first_keycode, defines the symbols for the specified number of keycodes. The symbols for keycodes outside this range remained unchanged. The number of elements in the keysyms list must be a multiple of keysyms_per_keycode. Otherwise, XChangeKeyboardMapping generates 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. Otherwise, it generates a BadValue error. In addition, the following expression must be less than or equal to max_keycode as returned in the connection setup (else a BadValue error).

first_keycode + (num_codes / keysyms_per_keycode) - 1

The KeySym number N, counting from zero, for keycode K has an index, counting from zero, of the following in keysyms:

(K - first_keycode) * keysyms_per_keycode + N

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

There is no requirement that the X server interpret this mapping. It is merely stored for reading and writing by clients.

XChangeKeyboardMapping can generate BadAlloc, BadLength, and BadValue errors.

The next four functions make use of the XModifierKeymap data structure.

```
typedef struct {
        int max_keypermod;          /* This server's max number of keys per modifier */
        KeyCode *modifiermap;       /* An 8 by max_keypermod array of the modifiers */
} XModifierKeymap;
```

To create an XModifierKeymap structure, use XNewModifierMap.

XModifierKeymap *XNewModifierMap(*max_keys_per_mod*)
    int *max_keys_per_mod*;

max_keys_per_mod
    Specifies the maximum number of keycodes assigned to any of the modifiers in the map.

The XNewModifierMapping function returns a XModifierKeymap structure.

To add a new entry to an XModifierKeymap structure, use XInsertModifiermapEntry.

```
XModifierKeymap *XInsertModifiermapEntry(modmap, keysym_entry, modifier)
    XModifierKeymap *modmap;
    KeyCode keysym_entry;
    int modifier;
```

modmap	Specifies a pointer to the XModifierKeymap structure.
keysym_entry	Specifies the keysyms.
modifier	Specifies the modifier.

The XInsertModifiermapEntry function add the specified keycode to the set that controls the specified modifier and returns the resulting XModifierKeymap structure (expanded as needed).

To delete an entry from an XModifierKeymap structure, use XDeleteModifiermapEntry.

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

modmap	Specifies a pointer to the XModifierKeymap structure.
keysym_entry	Specifies the keysyms.
modifier	Specifies the modifier.

The XDeleteModifiermapEntry function deletes the specified keycode from the set that controls the specified modifier and returns the resulting XModifierKeymap structure.

To destroy an XModifierKeymap structure, use XFreeModifierMap.

```
XFreeModifierMap(modmap)
     XModifierKeymap *modmap;
```

modmap	Specifies a pointer to the XModifierKeymap structure.

The XFreeModifierMapping function frees the specified XModifierKeymap structure.

To set which keycodes are to be used as modifiers, use XSetModifierMapping.

```
int XSetModifierMapping(display, modmap)
     Display *display;
     XModifierKeymap *modmap;
```

display	Specifies the connection to the X server.
modmap	Specifies a pointer to the XModifierKeymap structure.

The XSetModifierMapping function specifies the keycodes of the keys, if any, that are to be used as modifiers. A zero value means that no key should be used. No two arguments can have the same nonzero keycode value. Otherwise, XSetModifierMapping generates a BadValue error.

There are eight modifiers, and the modifiermap member of the XModifierKeymap structure contains eight sets of max_keypermod keycodes, one for each modifier in the order Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, and Mod5. Only nonzero keycodes have meaning in each set, and nonzero keycodes are ignored. In addition, all of the nonzero keycodes must be in the range specified by min_keycode and max_keycode in the Display structure. Otherwise, XSetModifierMapping generates a BadValue error. No keycode may appear twice in the entire map. Otherwise, it generates a BadValue error.

An X server can impose restrictions on how modifiers can be changed, for example, if certain keys do not generate up transitions in hardware or if multiple modifier keys are not supported. If some such restriction is violated, the status reply is MappingFailed, and none of the modifiers are changed. If the new keycodes specified for a modifier differ from those currently defined and any (current or new) keys for that modifier are in the logically down state, the status reply is MappingBusy, and none of the modifiers are changed. XSetModifierMapping generates a MappingNotify event on a MappingSuccess status.

XSetModifierMapping can generate BadAlloc and BadValue errors.

To obtain the keycodes that are to be used as modifiers, use XGetModifierMapping.

```
XModifierKeymap *XGetModifierMapping(display)
    Display *display;
```

display	Specifies the connection to the X server.

The XGetModifierMapping function returns a newly created XModifierKeymap structure that contains the keys being used as modifiers. The structure should be freed after use with XFreeModifierMapping. If only zero values appear in the set for any modifier, that modifier is disabled.

7.10. Screen Saver Control

Xlib provides functions with which you can set, force, activate, or reset the screen saver as well as obtain the current screen saver values.

To set the screen saver, use XSetScreenSaver.

```
XSetScreenSaver(display, timeout, interval, prefer_blanking, allow_exposures)
    Display *display;
    int timeout, interval;
    int prefer_blanking;
    int allow_exposures;
```

display	Specifies the connection to the X server.
timeout	Specifies the timeout, in seconds, until the screen saver turns on.
interval	Specifies the interval between screen saver invocations.
prefer_blanking	Specifies whether to enable screen blanking. Possible values are DontPreferBlanking, PreferBlanking, or DefaultBlanking.
allow_exposures	Specifies the current screen save control values. Possible values are DontAllowExposures, AllowExposures, or DefaultExposures.

Timeout and interval are specified in seconds. A timeout of 0 disables the screen saver, while a timeout of -1 restores the default. Other negative values generate a BadValue error. If the timeout value is nonzero, the function enables the screen saver. An interval of 0 disables the random pattern motion. If no input from devices (keyboard, mouse, and so on) is generated once the screen saver is enabled, for the specified number of timeout seconds, the screen saver is activated.

For each screen, if blanking is preferred and the hardware supports video blanking, the screen will simply go blank. Otherwise, if either exposures are allowed or the screen can be regenerated without sending exposure events to clients, the screen is tiled with the root window background tile at randomly re-origined each interval minutes. Otherwise, the state of the screens do not change, and the screen saver is not activated. The screen saver is deactivated and all screen states are restored at the next keyboard or pointer input or at the next call to XForceScreenSaver with mode ScreenSaverReset.

If the server-dependent screen saver method supports periodic change, the interval argument serves as a hint about how long the change period should be, and zero hints that no periodic change should be made. Examples of ways to change the screen include scrambling the colormap periodically, moving an icon image about the screen periodically, or tiling the screen with the root window background tile, randomly reoriginated periodically.

XSetScreenSaver can generate a BadValue error.

To force the screen saver on or off, use XForceScreenSaver.

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

display Specifies the connection to the X server.

mode Specifies the mode that is to be applied. XForceScreenSaver applies the specified mode to the screen saver. The possible modes are ScreenSaverActive or ScreenSaverReset.

If the specified mode is ScreenSaverActive and the screen saver currently is deactivated, the screen saver is activated, even if the screen saver had been disabled with a timeout of zero. If the specified mode is ScreenSaverReset and the screen saver currently is enabled, the screen saver is deactivated (if it was activated), and the activation timer is reset to its initial state (as if device input had been received).

XForceScreenSaver can generate a BadValue error.

To activate the screen saver, use XActivateScreenSaver.

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

display Specifies the connection to the X server.

To reset the screen saver, use XResetScreenSaver.

```
XResetScreenSaver(display)
    Display *display;
```

display Specifies the connection to the X server.

To get the current screen saver values, use XGetScreenSaver.

```
XGetScreenSaver(display, timeout_return, interval_return, prefer_blanking_return,
        allow_exposures_return)
    Display *display;
    int *timeout_return, *interval_return;
    int *prefer_blanking_return;
    int *allow_exposures_return;
```

display Specifies the connection to the X server.

timeout_return Returns the timeout, in minutes, until the screen saver turns on.

interval_return Returns the interval between screen saver invocations.

prefer_blanking_return
Returns the current screen blanking preference: DontPreferBlanking, PreferBlanking, or DefaultBlanking.

allow_exposures_return
Returns the current screen save control value: DontAllowExposures, AllowExposures, or DefaultExposures.

7.11. Controlling Host Access

This Section discusses how to:

- Add, get, or remove hosts from the access control list
- Change, enable, or disable access

X does not provide any protection on a per-window basis. If you find out the resource ID of a resource, you can manipulate it. To provide some minimal level of protection, however, connections are only permitted from machines you trust. This is adequate on single user workstations, but obviously breaks down on timesharing machines. While provisions exist in the X protocol for proper connection authentication, the lack of a standard authentication server leaves us with only host level access control. Currently, both DECnet and TCP domains are defined.

The initial set of hosts allowed to open connections consists of:

- The host the window system is running on.
- On UNIX-based systems, each host is listed in the /etc/X?.hosts file. The '?' indicates the number of the display. This file should consist of host names separated by newlines. DECnet nodes must terminate in "::" to distinguish them from internet hosts.

If a host is not in the access control list when the access control mechanism is enabled and if the host attempts to establish a connection, the server refuses the connection. To change the access list, the client must reside on the same host as the server and/or must have been granted permission in the initial authorization at connection setup. The initial access control list can be specified by providing a file that the server can read at startup and reset time.

Note

> Servers also can implement other access control policies in addition to or in place of this hosts access facility. The name and format of the information in this file is operating system specific. For further information about other access control implementations, see *X Window System Protocol, X Version 11.*

7.11.1. Adding, Getting, or Removing Hosts

Xlib provides functions with which you can add, get, or remove hosts. All the host access control functions use the XHostAddress structure. The elements in this structure are:

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

family	Specifies which protocol address family to use (for example, TCP/IP or DECnet). The family symbols are defined in <X11/X.h>.
length	Specifies the length of the address in bytes.
address	Specifies a pointer to the address.

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 2 bytes long. The first byte contains the least significant 8 bits of the node number. The second byte contains the most significant 2 bits of the node number in the least significant 2 bits of the byte and the area in the most significant 6 bits of the byte.

To add a single host, use XAddHost.

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

display	Specifies the connection to the X server.
host	Specifies the network address of the host machine.

The XAddHost function adds the specified host to the access control list for that display. The display hardware must be on the same host as the program issuing the command.

XAddHost can generate BadAlloc and BadValue errors.

To add multiple hosts at one time, use XAddHosts.

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

display	Specifies the connection to the X server.
hosts	Specifies each host that is to be added.
num_hosts	Specifies the number of hosts.

The XAddHosts function adds each specified host to the access control list for that display. The display hardware must be on the same host as the program issuing the command.

XAddHosts can generate BadAlloc and BadValue errors.

To obtain a host list, use XListHosts.

```
XHostAddress *XListHosts(display, nhosts_return, state_return)
    Display *display;
    int *nhosts_return;
    Bool *state_return;
```

display	Specifies the connection to the X server.
nhosts_return	Returns the number of hosts currently in the access control list.
state_return	Returns the state of the access control (enabled or disabled).

The XListHosts function returns the current access control list as well as whether the use of the list at connection setup was enabled or disabled. XListHosts allows a program to find out what machines can make connections. It also returns a pointer to a list of host structures that were allocated by the routine. When it no longer is needed, this memory should be freed by calling XFree. (See Section 2.4 for further information.)

To remove a single host, use XRemoveHost.

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

display	Specifies the connection to the X server.
host	Specifies the network address of the host machine.

The XRemoveHost function removes the specified host from the access control list for that display. The display hardware must be on the same host as the client process. If you remove your machine from the access list, you can no longer connect to that server, and this operation cannot be reversed short of resetting the server.

XRemoveHost can generate BadAlloc and BadValue errors.

To remove multiple hosts at one time, use XRemoveHosts.

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

display	Specifies the connection to the X server.
hosts	Specifies each host that is to be added.
num_hosts	Specifies the number of hosts.

The XRemoveHosts function removes each specified host from the access control list for that display. The display hardware must be on the same host as the client process. The XRemoveHosts function removes each specified host from the access control list for that display. The display hardware must be on the same host as the client process. If you remove your machine from the access list, you can no longer connect to that server, and this operation cannot be reversed short of resetting the server.

XRemoveHosts can generate BadAlloc and BadValue errors.

7.11.2. Changing, Enabling, or Disabling Access Control

Xlib provides functions with which you can enable, disable, or change access control.

For these functions to execute successfully, the client application must reside on the same host as the X server, and/or have been given permission in the initial authorization at connection setup.

To change access control, use XSetAccessControl.

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

display	Specifies the connection to the X server.
mode	Specifies whether you want to change the access control to enable or disable. EnableAccess enables host access control or DisableAccess disables host access control.

The XSetAccessControl function either enables or disables the use of the access control list at connection setups.

XSetAccessControl can generate BadAccess and BadAlloc errors.

To enable access control, use XEnableAccessControl.

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

display	Specifies the connection to the X server.

The XEnableAccessControl function enables the use of the access control list at connection setups.

XEnableAccessControl can generate a BadAccess error.

To disable access control, use XDisableAccessControl.

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

display	Specifies the connection to the X server.

The XDisableAccessControl function disables the use of the access control list at connection setups.

XDisableAccessControl can generate a BadAccess error.

Chapter 8

Events and Event-Handling Functions

A client application communicates with the X server through the connection you establish with the XOpenDisplay function. It is over this connection that a client application sends "requests" to the X server. These requests are made by the Xlib functions that are called in the client application. The X server sends back to the client application either "replies" or events. Most requests made by Xlib functions do not generate replies. Some requests generate multiple replies. Numerous Xlib functions cause the X server to generate events. In addition, the user's typing or moving the pointer can generate events asynchronously.

This Chapter begins with a discussion of the following topics associated with events:

- Event types
- Event structures
- Event mask
- Event processing

The Chapter continues with a discussion of the Xlib functions you can use to:

- Select events
- Handle the output buffer and the event queue
- Select events from the event queue
- Send and get events
- Handle error events

Note

Some toolkits use their own event-handling routines. Also, some toolkits do not allow you to interchange these event-handling routines with those in the Xlib library. See the document supplied with your toolkit for further information.

Most applications simply are event loops. That is, they wait for an event, decide what to do with it, execute some amount of code, which, in turn, results in changes to the display, and then wait for the next event.

8.1. Event Types

An event is data generated asynchronously by the X server as a result of some device activity, or as side effects of a request sent by an Xlib function. Device-related events propagate from the source window to ancestor windows until some client application has selected that event type, or until the event is explicitly discarded. The X server generally sends an event to a client application only if the client has specifically asked to be informed of that event type, usually by calling the XSelectInput Xlib function. However, KeymapNotify events are always sent. The mask can also be set when you create a window or by changing the window's event_mask. You can also mask out events that would propagate to ancestor windows by manipulating the do_not_propagate mask of the window's attributes.

The event type describes a specific event generated by the X server. For each event type, there is a corresponding constant name defined in <X11/X.h>. When referring to an event type, this manual uses the constant name defined in this file. It is often useful to group one or more event types into logical categories. For example, exposure processing refers to the processing that occurs for the exposure events Expose, GraphicsExpose, and NoExpose.

The following table lists the event category and its associated event type or types. The processing associated with these events is discussed in Section 8.4.

Event Category	Event Type
Keyboard events	KeyPress, KeyRelease
Pointer motion events	ButtonPress, s.PN ButtonRelease , MotionNotify
Window crossing events	EnterNotify, LeaveNotify
Input focus events	FocusIn, FocusOut
Key map state notification event	KeymapNotify
Exposure events	Expose, GraphicsExpose, NoExpose
Structure control events	CirculateRequest, ConfigureRequest, MapRequest, ResizeRequest
Window state notification events	CirculateNotify, ConfigureNotify, CreateNotify, DestroyNotify, GravityNotify, MapNotify, MappingNotify, ReparentNotify, UnmapNotify, VisibilityNotify
Color map state notification event	ColormapNotify
Client communication events	ClientMessage, PropertyNotify, SelectionClear, SelectionNotify, SelectionRequest

8.2. Event Structures

Each event type has a corresponding structure declared in <X11/Xlib.h>. All event structures have the following members:

type	Set to the event type constant name that uniquely identifies it. For example, when the X server reports a GraphicsExpose event to a client application, it sends an XGraphicsExposeEvent structure with the type member set to GraphicsExpose.
display	Set to a pointer to the display the event was read on.
send_event	Set to True if the event came from an XSendEvent request.
serial	Set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value.

The X server may send events at any time in the input stream, even between the time your client application sends a request and receives a reply. Xlib stores in an event queue any events received while waiting for a reply for later use. Xlib also provides functions that allow you to check events in the event queue. (These are discussed in Section 8.6.)

In addition to the individual structures declared for each event type, there is also a generic XEvent structure. The XEvent structure is a union of the individual structures declared for each event type. Once you determine the event type, use the structures declared in <X11/Xlib.h> when making references to it in a client application. All events contain a "type" member that determines the format of the information. Depending on the type, you should access elements of each event by using the XEvent union.

8.3. Event Mask

Clients select event reporting of most events relative to a window. To do this, you pass an event mask to an Xlib event-handling function that takes an event_mask argument. The bits of the event mask are defined in <X11/X.h>. Each bit in the event mask maps to an event mask name. The event mask name describes the event or events you want the X server to return to a client application. When referring to a specific event mask, this manual uses the constant name defined in this file.

Most events are not reported to clients when they are generated, unless the client has specifically asked for them. GraphicsExpose and NoExpose, however, are reported, by default, as a result of XCopyPlane and XCopyArea, unless the client suppresses them by setting graphics_expose in the GC to False. See Section 6.2 for further information. SelectionClear, SelectionRequest, SelectionNotify or ClientMessage cannot be masked, but they generally are only sent to clients cooperating with selections. See Section 4.4 for further information. MappingNotify is always sent to clients when the keyboard mapping is changed.

The following table lists the event mask constants you can pass to the event_mask argument and the circumstances in which you would want to specifies the event mask.

Event Mask	Circumstances
NoEventMask	No events wanted
KeyPressMask	Keyboard down events wanted
KeyReleaseMask	Keyboard up events wanted
ButtonPressMask	Pointer button down events wanted
ButtonReleaseMask	Pointer button up events wanted
EnterWindowMask	Pointer window entry events wanted
LeaveWindowMask	Pointer window leave events wanted
PointerMotionMask	Pointer motion events wanted
PointerMotionHintMask	Pointer motion hints wanted
Button1MotionMask	Pointer motion while button 1 down
Button2MotionMask	Pointer motion while button 2 down
Button3MotionMask	Pointer motion while button 3 down
Button4MotionMask	Pointer motion while button 4 down
Button5MotionMask	Pointer motion while button 5 down
ButtonMotionMask	Pointer motion while any button down
KeymapStateMask	Any keyboard state change wanted
ExposureMask	Any exposure wanted
VisibilityChangeMask	Any change in visibility wanted
StructureNotifyMask	Any change in window structure wanted
ResizeRedirectMask	Redirect resize of this window
SubstructureNotifyMask	Substructure notification wanted
SubstructureRedirectMask	Redirect substructure of window
FocusChangeMask	Any change in input focus wanted
PropertyChangeMask	Any change in property wanted
ColormapChangeMask	Any change in colormap wanted
OwnerGrabButtonMask	Automatic grabs should activate when owner_events is True

8.4. Event Processing

The event types reported to a client application during event processing depend on which event masks you pass to the event_mask argument of the XSelectInput function (Section 8.5). For some event masks, there is a one-to-one correspondence between the event mask constant and the event type constant. For example, if you pass the event mask ButtonPressMask, the X server sends back only ButtonPress events. Most events contain a time member that is the time at which an event occurred.

In other cases, one event mask constant can map to several event type constants. For example, if you pass the event mask SubstructureNotifyMask, the X server can send back CirculateNotify, ConfigureNotify, CreateNotify, DestroyNotify, GravityNotify, MapNotify, ReparentNotify, or UnmapNotify events.

In another case, two event mask constants map to one event type constant. For example, if you pass the event mask PointerMotionMask or PointerMotionHintMask the X server sends back a MotionNotify event.

The following table lists the event mask, its associated event type or types, and the structure name associated with the event type. Some of these structures actually are typedef's to a generic structure that is shared between two event types. Note that the letters N.A. appear in columns for which the information is not applicable.

Event Mask	Event Type	Structure	Generic Structure
ButtonMotionMask	MotionNotify	XPointerMovedEvent	XMotionEvent
Button1MotionMask			
Button2MotionMask			
Button3MotionMask			
Button4MotionMask			
Button5MotionMask			
ButtonPressMask	ButtonPress	XButtonPressedEvent	XButtonEvent
ButtonReleaseMask	ButtonRelease	XButtonReleasedEvent	XButtonEvent
ColormapChangeMask	ColormapNotify	XColormapEvent	
EnterWindowMask	EnterNotify	XEnterWindowEvent	XCrossingEvent
LeaveWindowMask	LeaveNotify	XLeaveWindowEvent	
ExposureMask	Expose	XExposeEvent	
	GraphicsExpose	XGraphicsExposeEvent	
	NoExpose	XNoExposeEvent	
FocusChangeMask	FocusIn	XFocusInEvent	XFocusChangeEvent
	FocusOut	XFocusOutEvent	XFocusChangeEvent
KeymapStateMask	KeymapNotify	XKeymapEvent	
KeyPressMask	KeyPress	XKeyPressedEvent	XKeyEvent
	KeyRelease	XKeyReleasedEvent	XKeyEvent
OwnerGrabButtonMask	N.A.	N.A.	
PointerMotionMask	MotionNotify	XPointerMovedEvent	XMotionEvent
PointerMotionHintMask			
PropertyChangeMask	PropertyNotify	XPropertyEvent	
ResizeRedirectMask	ResizeRequest	XResizeRequestEvent	
StructureNotifyMask	CirculateNotify	XCirculateEvent	
	ConfigureNotify	XConfigureEvent	
	DestroyNotify	XDestroyWindowEvent	
	GravityNotify	XGravityEvent	
	MapNotify	XMapEvent	

Event Mask	Event Type	Structure	Generic Structure
	ReparentNotify	XReparentEvent	
	UnmapNotify	XUnmapEvent	
SubstructureNotifyMask	CirculateNotify	XCirculateEvent	
	ConfigureNotify	XConfigureEvent	
	CreateNotify	XCreateWindowEvent	
	DestroyNotify	XDestroyWindowEvent	
	GravityNotify	XGravityEvent	
	MapNotify	XMapEvent	
	ReparentNotify	XReparentEvent	
	UnmapNotify	XUnmapEvent	
SubstructureRedirectMask	CirculateRequest	XCirculateRequestEvent	
	ConfigureRequest	XConfigureRequestEvent	
	MapRequest	XMapRequestEvent	
N.A.	ClientMessage	XClientMessageEvent	
N.A.	MappingNotify	XMappingEvent	
N.A.	SelectionClear	XSelectionClearEvent	
N.A.	SelectionNotify	XSelectionEvent	
N.A.	SelectionRequest	XSelectionRequestEvent	
VisibilityChangeMask	VisibilityNotify	XVisibilityEvent	

The Sections below describe the processing that occurs when you pass the different event masks to XSelectInput (Section 8.5). The Sections are organized according to these processing categories:

- Keyboard and pointer event processing
- Window crossing event processing
- Input focus event processing
- Keymap state notification event processing
- Exposure event processing
- Window state notification event processing
- Structure control event processing
- Colormap state notification event processing
- Client communication event processing

The processing descriptions include explanations of the structure or structures associated with the event. All the event structures contain the members type and display, which were discussed in Section 8.2. Thus, the explanations for these members are not repeated in the following Sections.

8.4.1. Keyboard and Pointer Event Processing

This Section discusses the event processing that occurs when a pointer button is pressed and when the keyboard events KeyPress and KeyRelease and the pointer motion events ButtonPress, ButtonRelease and MotionNotify are generated.

8.4.1.1. Pointer Button Specific Processing

The following describes the event processing that occurs when a pointer button press is processed with the pointer in some window w and when no active pointer grab is in progress.

The X server searches the ancestors of w from the root down, looking for a passive grab to activate. If no matching passive grab on the button exists, the X server automatically starts an active grab for the client receiving the event and sets the last-pointer-grab time to the current server time. The effect is essentially equivalent to an XGrabButton with these client passed arguments:

w	The event window
event_mask	The client's selected pointer motion events on the event window.
pointer_mode	GrabModeAsync.
keyboard_mode	GrabModeAsync.
owner_events	True, if the client has selected OwnerGrabButtonMask on the event window; otherwise, False.
confine_to	None.
cursor	None.

The active grab is automatically terminated when the logical state of the pointer has all buttons released. Clients can modify the active grab by calling XUngrabPointer and XChangeActivePointerGrab.

8.4.1.2. Common Keyboard and Pointer Event Processing

This Section discusses the processing that occurs for the keyboard events KeyPress and KeyRelease and the pointer motion events ButtonPress, ButtonRelease, and MotionNotify. See Chapter 10 for information about the keyboard event handling utility functions provided in XLib.

The X server can report KeyPress KeyRelease events to clients wanting information about when a key is pressed and KeyRelease events to clients wanting information about when a key logically changes state. Note that these events are generated for all keys, even those mapped to modifier bits. The X server reports ButtonPress or ButtonRelease events to clients wanting information about buttons that logically change state.

The X server reports MotionNotify events to clients wanting information about when the pointer logically moves. The X server generates this event whenever the pointer changes state, that is, whenever the pointer is moved and the pointer motion begins and ends in the window. The granularity of MotionNotify events is not guaranteed, but a client that selects this event type is guaranteed to receive at least one event when the pointer moves and then rests.

The generation of the logical changes may lag the physical changes if device event processing is frozen.

To receive KeyPress, KeyRelease, ButtonPress, and ButtonRelease events in a client application, you pass a window ID and KeyPressMask, KeyReleaseMask, ButtonPressMask, and ButtonReleaseMask as the event_mask arguments to XSelectInput.

To receive MotionNotify events in a client application, you pass a window ID and one or more of the following event masks as the event_mask argument to XSelectInput:

- Button1MotionMask–Button5MotionMask

  The client application receives MotionNotify events only when one or more of the specified buttons is pressed.

- ButtonMotionMask

  The client application receives MotionNotify events only when at least one button is pressed.

- PointerMotionMask

  The client application receives MotionNotify events independent of the state of the pointer buttons.
- PointerMotionHint

  If PointerMotionHintMask is selected, the X server is free to send only one MotionNotify event (with the is_hint member of the XPointerMovedEvent structure set to NotifyHint) to the client for the event window, until either the key or button state changes, or the pointer leaves the event window, or the client calls the XQueryPointer or XGetMotionEvents functions. The server still may send MotionNotify events without is_hint set to NotifyHint.

The source of the event is the smallest window containing the pointer. The window used by the X server to report these events depends on its position in the window hierarchy and whether any intervening window prohibits the generation of these events. The X server searches up the window hierarchy, starting with the source window, until it locates the first window specified by a client as having an interest in these events. If one of the intervening windows has its do_not_propagate_mask set to prohibit generation of the event type, the event of those types will be suppressed. Clients can modify the actual window used for reporting by performing active grabs, and, in the case of keyboard events, by using the focus window. See Chapter 7 for a discussion of the XGrabPointer, XGrabKeyboard, and XSetInputFocus functions.

The structures associated with these events are XKeyPressedEvent, XKeyReleasedEvent, XButtonPressedEvent, XButtonReleasedEvent, and XPointerMovedEvent. The window the event is in is called the event window. These structures have the following common members: window, root, subwindow, time, x, y, x_root, y_root, state, and same_screen.

The window member is set to the window ID of the window on which the event was generated and is referred to as the event window. This is the window used by the X server to report the event, as long as the conditions discussed in the previous paragraph are met. The root member is set to the window ID of the source window's root window. The x_root and y_root members are set to the pointer's coordinates relative to the root window's origin at the time of the event.

The same_screen member is set to indicate whether the event window is on the same screen as the root window and can be either True or False. If True, the event and root windows are on the same screen. If False, the event and root windows are not on the same screen.

If the source window is an inferior of the event window, the subwindow member of the structure is set to the child of the event window that is an ancestor of or is the source member. Otherwise, the X server sets the subwindow member to None. The time member is set to the time when the event was generated and is expressed in milliseconds since the server reset.

If the event window is on the same screen as the root window, the x and y members are set to the coordinates relative to the event window's origin. Otherwise, these members are set to zero.

The state member is set to indicate the logical state of the pointer buttons and modifier keys just prior to the event. For the state of the pointer buttons, the X server can set this member to the bitwise inclusive OR of one or more of the button or modifier key masks: Button1Mask, Button2Mask, Button3Mask, Button4Mask, Button5Mask, ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask.

Each of these structures also has a member that indicates the detail. For the XKeyPressedEvent and XKeyReleasedEvent structures, this member is called keycode. It is set to a number that represents a physical key on the keyboard. The keycode is an arbitrary representation for any key on the keyboard. See Chapter 10 for more information on the keycode.

For the XButtonPressedEvent and XButtonReleasedEvent structures, this member is called button. It represents the pointer buttons that changed state and can be set to the bitwise inclusive OR of one or more of these button names: Button1, Button2, Button3, Button4, Button5. For the XPointerMovedEvent structure, this member is called is_hint. It can be set to one of these constants: NotifyNormal or NotifyHint.

8.4.2. Window Entry/Exit Event Processing

This Section describes the processing that occurs for the window crossing events EnterNotify and LeaveNotify. If a pointer motion or a window hierarchy change causes the pointer to be in a different window than before, the X server reports EnterNotify or LeaveNotify events to clients who have selected for these events. All EnterNotify and LeaveNotify events caused by a hierarchy change are generated after any hierarchy event (UnmapNotify, MapNotify, ConfigureNotify, GravityNotify, CirculateNotify) caused by that change, but the ordering of EnterNotifyand LeaveNotify events with respect to FocusOut, VisibilityNotify, and Expose events is not constrained by the X protocol.

This contrasts with MotionNotify events, which are also generated when the pointer moves, but the pointer motion begins and ends in a single window. An EnterNotify or LeaveNotify event may also be generated when some client application calls XChangeActivePointerGrab, XGrabKeyboard, XGrabPointer, and XUngrabPointer.

To receive EnterNotify events in a client application, you must pass a window ID and EnterWindowMask as the event_mask argument to XSelectInput. Likewise, to receive LeaveNotify events, you pass the window ID and LeaveWindowMask.

The members of the XEnterWindowEvent and XLeaveWindowEvent structures associated with these events are window, root, subwindow, time, x, y, x_root, y_root, mode, detail, same_screen, focus, and state. The pointer position reported in the event is always the final position, not the initial position of the pointer.

The window member is set to the window ID of the window on which the EnterNotify or LeaveNotify event was generated and is referred to as the event window. This is the window used by the X server to report the event, and is relative to the root window on which the event occurred. The root member is the root window for this position and is set to the window ID of the root window on which the event occurred.

In a LeaveNotify event, if a child of the event window contains the initial position of the pointer, the subwindow component is set to that child. Otherwise, the X server sets the subwindow member to None. For an EnterNotify event, if a child of the event window contains the final pointer position, the subwindow component is set to that child. Otherwise, it is set to None.

The time member is set to the time when the event was generated and is expressed in milliseconds. The x and y members are set to the coordinates of the pointer position in the event window. This position is always the final position of the pointer, not the initial position of the pointer. If the event window is on the same screen as the root window, x and y are the pointer coordinates relative to the event window's origin. Otherwise, x and y are set to zero. The x_root and y_root members are set to the pointer's coordinates relative to the root window's origin at the time of the event.

The same_screen member is set to indicate whether the event window is on the same screen as the root window and can be either True or False. If True, the event and root windows are on the same screen. If False, the event and root windows are not on the same screen.

The focus member is set to indicate whether the event window is the focus window or an inferior of the focus window. The X server can set this member to either True or False. If True, the event window is the focus window or an inferior of the focus window. If False, the event window is not the focus window or an inferior of the focus window.

The state member is set to indicate the state of the pointer buttons and modifier keys just prior to the event. For the state of the pointer buttons, the X server can set this member to the bitwise inclusive OR of one or more of the button or modifier key masks: Button1Mask, Button2Mask, Button3Mask, Button4Mask, Button5Mask, ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask.

The mode member is set to indicate whether the events are normal events, pseudo-motion events when a grab activates, or pseudo-motion events when a grab deactivates. The X server can set this member to the corresponding constants NotifyNormal, NotifyGrab, or NotifyUngrab.

The detail member is set to indicate the notify detail and can be one of these constants: NotifyAncestor, NotifyVirtual, NotifyInferior, NotifyNonlinear, or NotifyNonlinearVirtual.

The following Sections discuss how the X server processes normal pointer motion events and pseudo motion events.

8.4.2.1. Normal Entry/Exit Event Processing

EnterNotify and LeaveNotify events are generated when the pointer moves from one window to another window. Normal events are identified by XEnterWindowEvent or XLeaveWindowEvent structures whose mode member is set to NotifyNormal.

- When the pointer moves from window A to window B, and A is an inferior of B, the X server generates:
  - A LeaveNotify event on window A with the detail member of the XLeaveWindowEvent structure set to NotifyAncestor.
  - A LeaveNotify event on each window between window A and window B exclusive, with the detail member of each XLeaveWindowEvent structure set to NotifyVirtual.
  - An EnterNotify event on window B with the detail member of the XEnterWindowEvent structure set to NotifyInferior.
- When the pointer moves from window A to window B, and B is an inferior of A, the X server generates:
  - A LeaveNotify event on window A with the detail member of the XLeaveWindowEvent structure set to NotifyInferior.
  - An EnterNotify event on each window between window A and window B exclusive, with the detail member of each XEnterWindowEvent structure set to NotifyVirtual.
  - An EnterNotify event on window B with the detail member of the XEnterWindowEvent structure set to NotifyAncestor.
- When the pointer moves from window A to window B, and window C is their least common ancestor, the X server generates:
  - A LeaveNotify event on window A with the detail member of the XLeaveWindowEvent structure set to NotifyNonlinear.
  - A LeaveNotify event on each window between window A and window C exclusive, with the detail member of each XLeaveWindowEvent structure set to NotifyNonlinearVirtual.
  - An EnterNotify event on each window between window C and window B exclusive, with the detail member of each XEnterWindowEvent structure set to NotifyNonlinearVirtual.
  - An EnterNotify event on window B with the detail member of the XEnterWindowEvent structure set to NotifyNonlinear.
- When the pointer moves from window A to window B on different screens, the X server:
  - Generates a LeaveNotify event on window A with the detail member of the XLeaveWindowEvent structure set to NotifyNonlinear.
  - If window A is not a root window, it generates a LeaveNotify event on each window above window A up to and including its root, with the detail member of each XLeaveWindowEvent structure set to NotifyNonlinearVirtual.

  - If window B is not a root window, it generates an EnterNotify event on each window from window B's root down to but not including window B, with the detail member of each XEnterWindowEvent structure set to NotifyNonlinearVirtual.
  - Generates an EnterNotify event on window B with the detail member of the XEnterWindowEvent structure set to NotifyNonlinear.

8.4.2.2. Grab and Ungrab Entry/Exit Event Processing

Pseudo-motion mode EnterNotify and LeaveNotify events are generated when a pointer grab activates or deactivates. Events in which the pointer grab activates are identified by XEnterWindowEvent or XLeaveWindowEvent structures whose mode member is set to NotifyGrab. Events in which the pointer grab deactivates are identified by XEnterWindowEvent or XLeaveWindowEvent structures whose mode member is set to NotifyUngrab. See the discussion of XGrabPointer discussed in Chapter 7.

- When a pointer grab activates, but after any initial warp into a confine_to window, and before generating any actual ButtonPress event that activates the grab, with G the grab_window for the grab and P the window the pointer is in, the X server:
  - Generates EnterNotify and LeaveNotify events (see Section 8.4.2.1) with the mode members of the XEnterWindowEvent and XLeaveWindowEvent structures set to NotifyGrab. These events are generated as if the pointer were to suddenly warp from its current position in P to some position in G. However, the pointer does not warp, and the X server uses the pointer position as both the initial and final positions for the events.
- When a pointer grab deactivates, but after generating any actual ButtonRelease event that deactivates the grab, with G the grab_window for the grab and P the window the pointer is in, the X server:
  - Generates EnterNotify and LeaveNotify events (see Section 8.4.2.1) with the mode members of the XEnterWindowEvent and XLeaveWindowEvent structures set to NotifyUngrab. These events are generated as if the pointer were to suddenly warp from some position in G to its current position in P. However, the pointer does not warp, and the X server uses the current pointer position as both the initial and final positions for the events.

8.4.3. Input Focus Events

This Section describes the processing that occurs for the input focus events FocusIn and FocusOut. The X server can report FocusIn or FocusOut events to clients wanting information about when the input focus changes. The input focus is where the keyboard input goes. The keyboard is always attached to some window, typically, the root window or a top-level window, which is called the focus window. The focus window and the position of the pointer determines the window that receives keyboard input. Clients may need to know when the input focus changes. This is often used to control highlighting of areas of the screen. See also the focus member enter/exit events in Section 8.4.2. This can occur when a client calls XGrabKeyboard and XUngrabKeyboard.

To receive FocusIn and FocusOut events in a client application, you pass a window ID and FocusChangeMask as the event_mask argument to XSelectInput.

The members of the XFocusInEvent and XFocusOutEvent structures associated with these events are window, mode, and detail. The window member is set to the window ID of the window on which the FocusIn or FocusOut event was generated. This is the window used by the X server to report the event. The mode member is set to indicate whether the focus events are normal focus events, while grabbed focus events, focus events when a grab activates, or focus events when a grab deactivates. The X server can set the mode member to the corresponding constants NotifyNormal, NotifyWhileGrabbed, NotifyGrab, or NotifyUngrab. The following Sections discuss how the X server processes normal focus events, while grabbed focus events, and grab activate/deactivate focus events.

All FocusOut events caused by a window unmap are generated after any UnmapNotify event, but the ordering of FocusOut events with respect to generated EnterNotify, LeaveNotify, VisibilityNotify, and Expose events is not constrained by the X protocol.

Depending on the event mode, the detail member is set to indicate the notify detail and can be one of these constants: NotifyAncestor, NotifyVirtual, NotifyInferior, NotifyNonlinear, NotifyNonlinearVirtual, NotifyPointer, NotifyPointerRoot, or NotifyDetailNone.

8.4.3.1. Normal and While Grabbed Focus Event Processing

Normal focus events are identified by XFocusInEvent or XFocusOutEvent structures whose mode member is set to NotifyNormal, while grabbed focus events are identified by XFocusInEvent or XFocusOutEvent structures whose mode member is set to NotifyWhileGrabbed. The X server processes normal focus and while grabbed focus events according to the following focus scenarios:

- When the focus moves from window A to window B, and A is an inferior of B with the pointer in window P, the X server:
  - Generates a FocusOut event on window A with the detail member of the XFocusOutEvent structure set to NotifyAncestor.
  - Generates a FocusOut event on each window between window A and window B exclusive, with the detail member of each XFocusOutEvent structure set to NotifyVirtual.
  - Generates a FocusIn event on window B with the detail member of the XFocusOutEvent structure set to NotifyInferior.
  - If window P is an inferior of window B, but window P is not window A or an inferior of window A, it generates a FocusIn event on each window below window B down to and including window P, with the detail member of each XFocusInEvent structure set to NotifyInferior.
- When the focus moves from window A to window B, and B is an inferior of A with the pointer in window P, the X server:
  - If window P is an inferior of window A, but P is not A, or an inferior of window B, or an ancestor of B, it generates a FocusOut event on each window from window P up to but not including window A (in that order), with the detail member of each XFocusOutEvent structure set to NotifyPointer.
  - Generates a FocusOut event on window A with the detail member of the XFocusOutEvent structure set to NotifyInferior.
  - Generates a FocusIn event on each window between window A and window B exclusive, with the detail member of each XFocusInEvent structure set to NotifyVirtual.
  - Generates a FocusIn event on window B with the detail member of the XFocusInEvent structure set to NotifyAncestor
- When the focus moves from window A to window B, and window C is their least common ancestor, and with the pointer in window P, the X server:
  - If window P is an inferior of window A, it generates a FocusOut event on each window from window P up to but not including window A, with the detail member of the XFocusOutEvent structure set to NotifyPointer.
  - Generates a FocusOut event on window A with the detail member of the XFocusOutEvent structure set to NotifyNonlinear.
  - Generates a FocusOut event on each window between window A and window C exclusive, with the detail member of each XFocusOutEvent structure set to NotifyNonlinearVirtual.

  - Generates a FocusIn event on each window between C and B exclusive, with the detail member of each XFocusInEvent structure set to NotifyNonlinearVirtual.
  - Generates a FocusIn event on window B with the detail member of the XFocusInEvent structure set to NotifyNonlinear.
  - If window P is an inferior of window B, it generates a FocusIn event on each window below window B down to and including window P, with the detail member of the XFocusInEvent structure set to NotifyPointer.
- When the focus moves from window A to window B on different screens with the pointer in window P, the X server:
  - If window P is an inferior of window A, it generates a FocusOut event on each window from window P up to but not including window A, with the detail member of each XFocusOutEvent structure set to NotifyPointer.
  - Generates a FocusOut event on window A with the detail member of the XFocusOutEvent structure set to NotifyNonlinear.
  - If window A is not a root window, it generates a FocusOut event on each window above window A up to and including its root, with the detail member of each XFocusOutEvent structure set to NotifyNonlinearVirtual.
  - If window B is not a root window, it generates a FocusIn event on each window from window B's root down to but not including window B, with the detail member of each XFocusInEvent structure set to NotifyNonlinearVirtual.
  - Generates a FocusIn event on window B with the detail member of each XFocusInEvent structure set to NotifyNonlinear.
  - If window P is an inferior of window B, it generates a FocusIn event on each window below window B down to and including window P, with the detail member of each XFocusInEvent structure set to NotifyPointer.
- You may have specified the focus window by passing PointerRoot (or None), when you called the function XSetInputFocus. (See Chapter 7.) When the focus moves from window A to PointerRoot (events sent to the window under the pointer) or None (discard), with the pointer in window P, the X server:
  - If window P is an inferior of window A, it generates a FocusOut event on each window from window P up to but not including window A, with the detail member of each XFocusOutEvent structure set to NotifyPointer.
  - Generates a FocusOut event on window A with the detail member of the XFocusOutEvent structure set to NotifyNonlinear.
  - If window A is not a root window, it generates a FocusOut event on each window above window A up to and including its root, with the detail member of each XFocusOutEvent structure set to NotifyNonlinearVirtual.
  - Generates a FocusIn event on the root window of all screens with the detail member of each XFocusInEvent structure set to NotifyPointerRoot (or NotifyDetailNone).
  - If the new focus is PointerRoot, it generates a FocusIn event on each window from window P's root down to and including window P, with the detail member of each XFocusInEvent structure set to NotifyPointerRoot.
- When the focus moves from PointerRoot (events sent to the window under the pointer) or None to window A, with the pointer in window P, the X server:
  - If the old focus is PointerRoot, it generates a FocusOut event on each window from window P up to and including window P's root, with the detail member of each XFocusOutEvent structure set to NotifyPointerRoot (in order).

   - Generates a FocusOut event on all root windows with the detail member of each XFocusOutEvent structure set to NotifyPointerRoot (or NotifyDetailNone).
   - If window A is not a root window, it generates a FocusIn event on each window from window A's root down to but not including window A, with the detail member of each XFocusInEvent structure set to NotifyNonlinearVirtual.
   - Generates a FocusIn event on window A with the detail member of the XFocusInEvent structure set to NotifyNonlinear.
   - If window P is an inferior of window A, it generates a FocusIn event on each window below window A down to and including window P, with the detail member of each XFocusInEvent structure set to NotifyPointer.
- When the focus moves from PointerRoot (events sent to the window under the pointer) to None (or vice versa), with the pointer in window P, the X server:
   - If the old focus is PointerRoot, it generates a FocusOut event on each window from window P up to and including window P's root, with the detail member of each XFocusOutEvent structure set to NotifyPointerRoot.
   - Generates a FocusOut event on all root windows with the detail member of each XFocusOutEvent structure set to either NotifyPointerRoot or NotifyDetailNone.
   - Generates a FocusIn event on all root windows with the detail member of each XFocusInEvent structure set to NotifyDetailNone or NotifyPointerRoot.
   - If the new focus is PointerRoot, it generates a FocusIn event on each window from window P's root down to and including window P, with the detail member of each XFocusInEvent structure set to NotifyPointerRoot.

8.4.3.2. Focus Events Generated by Grabs

Focus events in which the keyboard grab activates are identified by XFocusInEvent or XFocusOutEvent structures whose mode member is set to NotifyGrab. Focus events in which the keyboard grab deactivates are identified by XFocusInEvent or XFocusOutEvent structures whose mode member is set to NotifyUngrab. See the discussion of XGrabKeyboard in Chapter 7.

- When a keyboard grab activates but before generating any actual KeyPress event that activates the grab with G the grab_window and F the current focus, the X server:
   - Generates FocusIn and FocusOut events (as discussed in the previous Section), with the mode members of the XFocusInEvent and XFocusOutEvent structures set to NotifyGrab. These events are generated as if the focus were to change from F to G.
- When a keyboard grab deactivates but after generating any actual KeyRelease event that deactivates the grab with G the grab_window and F the current focus, the X server:
   - Generates FocusIn and FocusOut events (as discussed in the previous Section), with the mode members of the XFocusInEvent and XFocusOutEvent structures set to NotifyUngrab. These events are generated as if the focus were to change from G to F.

8.4.4. Keymap State Notification Event Processing

This Section discusses the processing that occurs for the keymap state notification event KeymapNotify. The X server reports KeymapNotify events to clients wanting information about changes in the keyboard state. To receive KeymapNotify events in a client application, you pass member is set to the bit vector of the keyboard. Each one bit indicates that the corresponding key is currently pressed. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N+7 with the least significant bit in the byte representing key 8N.

8.4.5. Exposure Event Processing

The X protocol does not guarantee to preserve the contents of window regions when the windows are obscured or reconfigured. Some implementations may preserve the contents of windows. Many other implementations will feel free to destroy the contents of windows when exposed. X expects client applications to assume the responsibility for restoring the contents of an exposed window region. (An exposed window region describes a formerly obscured window whose region or piece of a region becomes visible). Therefore, the X server sends exposure events describing the window and the region of the window that has been exposed. A trivial client application usually redraws the entire window. A more sophisticated client application redraws only the exposed region. The following Sections discuss the processing that occurs for the Expose, GraphicsExpose, and NoExpose exposure events.

8.4.5.1. Expose Event Processing

The X server can report Expose events to clients wanting information about when the contents of window regions have been lost. The circumstances in which the X server generates Expose events are not as definite as those for other events. However, the X server never generates Expose events on windows whose class you specified as InputOnly. The X server can generate Expose events when no valid contents are available for regions of a window and either the regions are visible, or the regions are viewable and the server is (perhaps newly) maintaining backing store on the window, or the window is not viewable but the server is (perhaps newly) honoring the window's backing-store attribute of Always or WhenMapped. The regions are decomposed into an (arbitrary) set of rectangles, and an Expose event is generated for each rectangle. The X server guarantees to report contiguously for any given window all of the regions exposed by some action that causes Exposure events, such as raising a window.

To receive Expose events in a client application, you pass a window ID and ExposureMask as the event_mask argument to XSelectInput.

The members of the XExposeEvent structure associated with this event are window, x, y, width, height, and count. The window member is set to the window ID of the exposed (damaged) window. The x and y members are set to the coordinates relative to the drawable's origin and indicate the upper-left corner of the rectangle. The width and height members are set to the size (extent) of the rectangle. The count member is set to the number of Expose events that are to follow. If count is set to 0 (zero), no more Expose events follow for this window. However, if count is set to nonzero, at least count Expose events and possibly more follow for this window. Simple applications that do not want to optimize redisplay by distinguishing between subareas of its window can just ignore all Expose events with nonzero counts and perform full redisplays on events with zero counts.

8.4.5.2. GraphicsExpose and NoExpose Event Processing

The X server can report GraphicsExpose events to clients wanting information about when a destination region could not be computed during a graphics request. Clients initiate a graphics request by calling XCopyArea or XCopyPlane. The X server generates this event whenever a destination region could not be computed due to an obscured or out-of-bounds source region. In addition, the X server guarantees to report contiguously all of the regions exposed by some graphics request (for example, copying an area of a drawable to a destination drawable).

The X server generates NoExpose events whenever a graphics request that might produce a GraphicsExpose event does not produce any. In other words, the client is really asking for a GraphicsExpose event but instead receives a NoExpose event.

To receive GraphicsExpose or NoExpose events in a client application, you must first set the graphics_exposures member of the XGCValues structure for the associated graphics context to True. You did this when you created the graphics context by calling XCreateGC. Or, you may have set the graphics_exposures member after creating the graphics context by calling XSetGraphicsExposures. See Chapter 5 for more information on these functions.

The structures associated with these event types are XGraphicsExposeEvent and XNoExposeEvent, and they have these common members: drawable, major_code, and minor_code. The drawable member is set to the drawable ID of the destination region on which the copy request was to be performed. The major_code member is set to the graphics request initiated by the client, and can be either X_CopyArea or X_CopyPlane. If X_CopyArea, a call to XCopyArea initiated the request. If X_CopyPlane, a call to XCopyPlane initiated the request. These constants are defined in <X11/Xproto.h>. The minor_code member, like the major_code member, indicates which graphics request was initiated by the client. However, the minor_code member is not defined by the core X protocol and will be zero in these cases, but may be used by an extension.

The XGraphicsExposeEvent structure has these additional members: x, y, width, height, and count. The x and y members are set to the coordinates relative to the drawable's origin and indicate the upper-left corner of the rectangle. The width and height members are set to the size (extent) of the rectangle. The count member is set to the number of GraphicsExpose events to follow. If count is set to 0 (zero), no more GraphicsExpose events follow for this window. However, if count is set to nonzero, at least that many, and possibly more, GraphicsExpose events are to follow for this window.

8.4.6. Window State Notification Event Processing

The following Sections discuss the processing that occurs for these window state notification events: CirculateNotify, ConfigureNotify, CreateNotify, DestroyNotify, GravityNotify, MapNotify, MappingNotify, ReparentNotify, UnmapNotify, and VisibilityNotify.

8.4.6.1. CirculateNotify Event Processing

The X server can report CirculateNotify events to clients wanting information about when a window changes its position in the stack. The X server generates this event type whenever a window is actually restacked as a result of a client application calling XCirculateSubwindows, XCirculateSubwindowsUp, or XCirculateSubwindowsDown.

To receive this event type in a client application, you pass the window ID and StructureNotifyMask as the event_mask argument to XSelectInput. You can also receive this event type by passing the parent window ID and SubstructureNotifyMask.

The members of the XCirculateEvent structure associated with this event are event, window, and place. The event member is set to the window ID of the window on which the CirculateNotify event was generated. This is the window used by the X server to report the event. The window member is set to the window ID of the window that was restacked. The place member is set to the window's position after the restack occurs, and is either PlaceOnTop or PlaceOnBottom. If PlaceOnTop, the window is now on top of all siblings. If PlaceOnBottom, the window is now below all siblings.

8.4.6.2. ConfigureNotify Event Processing

The X server can report ConfigureNotify events to clients wanting information about actual changes to a window's state, that is, size, position, border, stacking order. The X server generates this event type whenever one of the following configure window requests made by a client application actually completes:

- A window's size, position, border, and stacking order is reconfigured as a result of calling XConfigureWindow.

- The window's position in the stacking order is changed as a result of calling XLowerWindow, XRaiseWindow, or XRestackWindows.
- A window is moved as a result of calling XMoveWindow.
- A window's size is changed as a result of calling XResizeWindow.
- A window's size and location is changed as a result of calling XMoveResizeWindow.
- A window is mapped and its position in the stacking order is changed as a result of calling XMapRaised.
- A window's border width is changed as a result of calling XSetWindowBorderWidth.

To receive this event type in a client application, you pass the window ID and StructureNotifyMask as the event_mask argument to XSelectInput. You can also receive this event type by passing the parent window ID and SubstructureNotifyMask.

The members of the XConfigureEvent structure associated with this event are event, window, x, y, width, height, border_width, above, and override_redirect. The event member is set to the window ID of the window on which the ConfigureNotify event was generated. This is the window used by the X server to report the event. The window member is set to the window ID of the window whose size, position, border, and/or stacking order was changed.

The x and y members are set to the coordinates relative to the new parent window's origin and indicate the position of the upper-left outside corner of the window. The width and height members are set to the inside size of the window, not including the border. The border_width member is set to the width of the window's border, in pixels.

The above member is set to the window ID of the sibling window and is used for stacking operations. If the X server sets this member to None, the window whose state was changed is on the bottom of the stack with respect to sibling windows. However, if this member is set to the ID of a sibling window, the X server places the window whose state was changed on top of this sibling window.

The override_redirect member is set to the constant you specified for the override_redirect member of the XSetWindowAttributes structure when you created the window or changed its attributes. This constant is either True or False. Window manager clients normally should ignore this event if the override_redirect member is True.

8.4.6.3. CreateNotify Event Processing

The X server can report CreateNotify events to clients wanting information about creation of windows. The X server generates this event whenever a client application creates a window by calling XCreateWindow or XCreateSimpleWindow.

To receive this event type in a client application, you pass the window ID of the parent window and SubstructureNotifyMask as the event_mask argument to XSelectInput.

The members of the XCreateWindowEvent structure associated with this event are parent, window, x, y, width, height, border_width, and override_redirect. The parent member is set to the window ID of the created window's parent. The window member specifies the window ID of the created window. The x and y members are set to the created window's coordinates relative to the inside of the parent window's borders and indicate the position of the upper-left outside corner of the created window. The width and height members are set to the inside size of the created window not including the border, and are always a nonzero value. The border_width member is set to the width of the created window's border, in pixels. The override_redirect member is set to the constant you specified for the override_redirect member of the XSetWindowAttributes structure when you created the window or changed its attributes. This constant is either True or False. Window manager clients normally should ignore this event if the override_redirect member is True.

8.4.6.4. DestroyNotify Event Processing

The X server can report DestroyNotify events to clients wanting information about which windows are destroyed. The X server generates this event whenever a client application destroys a window by calling XDestroyWindow or XDestroySubwindows.

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 by the X protocol.

To receive this event type in a client application, you pass the window ID of the parent window and SubstructureNotifyMask as the event_mask argument to XSelectInput. You can also receive this event type by passing the window ID of the window and SubstructureNotifyMask.

The members of the XDestroyWindowEvent structure associated with this event are event and window. The event member is set to the window ID of the window on which the DestroyNotify event was generated. This is the window used by the X server to report the event. The window member is set to the window ID of the window that is destroyed.

8.4.6.5. GravityNotify Event Processing

The X server can report GravityNotify events to clients wanting information about when a window is moved because of a change in the size of its parent. The X server generates this event whenever a client application actually moves a child window as a result of resizing its parent by calling XConfigureWindow, XMoveResizeWindow, and XResizeWindow.

To receive this event type in a client application, you pass the window ID of the window and StructureNotifyMask as the event_mask argument to XSelectInput. You can also receive this event type by passing the window ID of the parent window and SubstructureNotifyMask.

The members of the XGravityEvent structure associated with this event are event, window, x, and y. The event member is set to the window ID of the window on which the GravityNotify event was generated. This is the window used by the X server to report the event. The window member is set to the window ID of the child window that was moved. The x and y members are set to the coordinates relative to the new parent window's origin and indicate the position of the upper-left outside corner of the window.

8.4.6.6. MapNotify Event Processing

The X server can report MapNotify events to clients wanting information about which windows are mapped. The X server generates this event type whenever a client application changes the window's state from unmapped to mapped by calling XMapWindow, XMapRaised, or XMapSubwindows.

To receive this event type, you pass the window ID of the window and StructureNotifyMask as the event_mask argument to XSelectInput. You can also receive this event type by passing the window ID of a parent window and SubstructureNotifyMask.

The members of the XMapEvent structure associated with this event are event, window, and override_redirect. The event member is set to the window ID of the window on which the MapNotify event was generated. This is the window used by the X server to report the event. The window member is set to the window ID of the window that was mapped. The override_redirect member is set to the constant you specified for the override_redirect member of the XSetWindowAttributes structure, when you created the window or changed its attributes. This constant is either True or False. Window manager clients normally should ignore this event if the override_redirect member is True, because these events usually are generated from pop-ups, which override structure control.

8.4.6.7. MappingNotify Event Processing

The X server reports MappingNotify events to all clients. There is no mechanism to express disinterest in this event. The X server generates this event type whenever a client application calls:

- XSetModifierMapping to indicate which keycodes are to be used as modifiers. The status reply must be MappingSuccess.
- XChangeKeyboardMapping to change the keyboard mapping.
- XSetPointerMapping to set the pointer mapping. The status reply must be Mapping-Success.

See Sections 7.8 and 7.9 for a discussion of these functions.

The members of the XMappingEvent structure associated with this event are window (not used and only present to aid certain toolkits), request, first_keycode, and count. The request member is set to indicate the kind of mapping change that occurred, and can be one of the constants MappingModifier, MappingKeyboard, MappingPointer. If MappingModifier, the specified keycodes are used as modifiers. If MappingKeyboard, the keyboard mapping is changed. If MappingPointer, the pointer button mapping is set. The first_keycode and count members are set only if the request member was set to MappingKeyboard. If this is the case, these members are set to numbers that indicate the range of altered keyboards. Thus, the number in first_keycode represents the first number in the range, and the number in count represents the last number in the range.

To update the client application's knowledge of the keyboard, you should call XRefreshKey-boardMapping. See Chapter 10 for a discussion of this function.

8.4.6.8. ReparentNotify Event Processing

The X server can report ReparentNotify events to clients wanting information about changing a window's parent. The X server generates this event whenever a client application calls XReparentWindow and the window is actually reparented.

To receive this event type in a client application, you pass the window ID of the old or the new parent window and SubStructureNotifyMask as the event_mask argument to XSelectInput. You can also receive this event type by passing the window ID and StructureNotifyMask.

The members of the XReparentEvent structure associated with this event are event, window, parent, x, y, and override_redirect. The event member is set to the window ID of the window on which the ReparentNotify event was generated and on which you requested notification by using XSelectInput. This is the window used by the X server to report the event. The window member is set to the window ID of the window that was reparented. The parent member is set to the window ID of the new parent window. The x and y members are set to the reparented window's coordinates relative to the new parent window's origin and define the upper-left outer corner of the reparented window. The override_redirect member is set to the constant you specified for the override_redirect member of the XSetWindowAttributes structure when you created the window or changed its attributes. This constant is either True or False. Window manager clients normally should ignore this event if the override_redirect member is True.

8.4.6.9. UnmapNotify Event Processing

The X server can report UnmapNotify events to clients wanting information about which windows are unmapped. The X server generates this event type whenever a client application changes the window's state from mapped to unmapped by calling XUnmapWindow or XUn-mapSubwindows.

To receive this event type, you pass the window ID and StructureNotifyMask as the event_mask argument to XSelectInput. You can also receive this event type by passing the window ID of the parent window and SubstructureNotifyMask.

The members of the XUnmapEvent structure associated with this event are event, window, and from_configure. The event member is set to the window ID of the window on which the UnmapNotify event was generated and on which you requested notification by using XSelectInput. This is the window used by the X server to report the event. The window member is set to the window ID of the window that was unmapped. The from_configure member is set to True if the event was generated as a result of a resizing of the window's parent when the window itself had a win_gravity of UnmapGravity.

8.4.6.10. VisibilityNotify Event Processing

The X server can report VisibilityNotify events to clients wanting any change in the visibility of the specified window. A region of a window is visible if someone looking at the screen can actually see it. The X server generates this event whenever the visibility changes state. However, this event is never generated for windows whose class is InputOnly. X ignores any subwindows in this computation.

All VisibilityNotify events caused by a hierarchy change are generated after any hierarchy event (UnmapNotify, MapNotify, ConfigureNotify, GravityNotify, CirculateNotify) caused by that change. Any VisibilityNotify event on a given window is generated before any Expose events on that window, but it is not required that all VisibilityNotify events on all windows be generated before all Expose events on all windows. The ordering of VisibilityNotify events with respect to FocusOut, EnterNotify, and LeaveNotify events is not constrained by the X protocol.

To receive this event type in a client application, you pass the window ID of the window and VisibilityChangeMask as the event_mask argument to XSelectInput.

The members of the XVisibilityEvent structure associated with this event are window and state. The window member is set to the window whose visibility state changes. The state member is set to the state of the window's visibility, and can be one of the constants VisibilityUnobscured, VisibilityPartiallyObscured, or VisibilityFullyObscured. The X server ignores all of a window's subwindows when determining the visibility state of the window and processes VisibilityNotify events according to the following:

- When the window changes state from partially or fully obscured or not viewable to viewable and completely unobscured, the X server generates the event with the state member of the XVisibilityEvent structure set to VisibilityUnobscured.
- When the window changes state from viewable and completely unobscured or from not viewable to viewable and partially obscured, the X server generates the event with the state member of the XVisibilityEvent structure set to VisibilityPartiallyObscured.
- When the window changes state from viewable and completely unobscured or viewable and partially obscured or from not viewable to viewable and fully obscured, the X server generates the event with the state member of the XVisibilityEvent structure set to VisibilityFullyObscured.

8.4.7. Structure Control Event Processing

The following Sections discuss the processing that occurs for these structure control events: CirculateRequest, ConfigureRequest, MapRequest, and ResizeRequest. These are only generated when clients have structure control enabled and are generally only of interest to window managers.

8.4.7.1. CirculateRequest Event

The X server can report CirculateRequest events to clients wanting information about when another client initiates a circulate window request on a specified parent window. The X server generates this event type whenever a client initiates a circulate window request on a parent window, and a window actually needs to be restacked. The client initiates a circulate window request on the parent window by calling XCirculateSubwindows, XCirculateSubwindowsUp, or XCirculateSubwindowsDown.

To receive this event type in a client application, you pass the window ID of the parent window and SubstructureRedirectMask as the event_mask argument to XSelectInput. In the future, the circulate window request for the specified window will not be not executed, and, thus, the window's position in the stack is not changed. For example, suppose a client application calls XCirculateSubwindowsUp to raise a specified window to the top of the stack. If you had selected SubstructureRedirectMask on the parent window, the X server reports to you a CirculateRequest event and does not raise the specified window to the top of the stack.

The members of the XCirculateRequestEvent structure associated with this event are parent, window, and place. The parent member is set to the window ID of the parent window. The window member is set to the window ID of the window to be restacked. The place member is set to what the new position in the stacking order should be, and is either PlaceOnTop or PlaceOnBottom. If PlaceOnTop, the window should be on top of all siblings. If PlaceOnBottom, the window should be below all siblings.

8.4.7.2. ConfigureRequest Event

The X server can report ConfigureRequest events to clients wanting information about when another client initiates a configure window request on a specified window. The configure window request attempts to reconfigure a window's size, position, border, and stacking order. The X server generates this event whenever a client initiates a configure window request on a window by calling XConfigureWindow, XLowerWindow, XRaiseWindow, XMapRaised, XMoveResizeWindow, XMoveWindow, XResizeWindow, XRestackWindows, or XSetWindowBorderWidth.

To receive this event type in a client application, you pass the window ID of the parent window and SubstructureRedirectMask as the event_mask argument to XSelectInput. It is generated when a ConfigureWindow request is issued on the window by another client. For example, suppose a client application calls XLowerWindow to lower a window. If you had selected SubstructureRedirectMask on the parent window, and if the override_redirect member of the XSetWindowAttributes structure associated with the specified window is set to False, the X server reports a ConfigureRequest event to you and does not lower the specified window.

The members of the XConfigureRequestEvent structure associated with this event are parent, window, x, y, width, height, border_width, above, detail, and value_mask. The parent member is set to the window ID of the parent window. The window member is set to the window ID whose size, position, border width, and/or stacking order is to be reconfigured. The x and y members are set to the coordinates relative to the parent window's origin and indicate the desired position of the upper-left outside corner of the reconfigured window. The width and height members are set to the desired inside size of the reconfigured window (not including the border) and are always a nonzero value. The border_width member is set to the desired width of the reconfigured window's border in pixels. The above member is set to the window ID of the sibling window and is used for stacking operations. If the X server sets this member to None, then the reconfigured window should be placed on the bottom of the stack with respect to sibling windows. However, if this member is set to the ID of a sibling window, the reconfigured window wants to be placed on top of this sibling window.

If not given in the request, the detail member is set to Above. This member could also be set to the constants Below, TopIf, BottomIf, or Opposite. The value_mask member is set to indicate which components were specified in the request. The value_mask and the corresponding values are reported as given in the request.

8.4.7.3. MapRequest Event

The X server can report MapRequest events to clients wanting information about another client's desire to map (place) windows. A window is considered mapped when a map window request completes. The X server generates this event whenever a client initiates a map window request on an unmapped window whose override_redirect member is set to False. Clients initiate map window requests by calling XMapWindow, XMapRaised, or XMapSubwindows.

To receive this event type in a client application, you pass the window ID of the parent window and SubstructureRedirectMask as the event_mask argument to XSelectInput. This means another client's attempts to map the window by calling one of the map window request functions will fail, and you will be sent a MapRequest instead. For example, suppose a client application calls XMapWindow to map a window. If you (usually your window manager) had selected SubstructureRedirectMask on the parent window, and if the override_redirect member of the XSetWindowAttributes structure associated with the specified window is set to False, the X server reports to you a MapRequest event and does not map the specified window. Thus, this event gives your window manager client the ability to control the placement of subwindows.

The members of the XMapRequestEvent structure associated with this event are parent and window. The parent member is set to the window ID of the parent window. The window member is set to the window ID of the window to be mapped.

8.4.7.4. ResizeRequest Event Processing

The X server can report ResizeRequest events to clients wanting information about another client's attempts to change the size of a window. The X server generates this event whenever some other client attempts to change the size of the specified window by calling XConfigureWindow, XResizeWindow, or XMoveResizeWindow.

To receive this event type in a client application, you pass a window ID and ResizeRedirect as the event_mask argument to XSelectInput. You will be sent the ResizeRedirect event, and any attempts to change the size by other clients will fail.

The members of the XResizeRequestEvent structure associated with this event are window, width, and height. The window member is set to the window ID of the window whose size another client attempted to change. The width and height members are set to the inside size of the window, not including the border.

8.4.8. Colormap State Notification Event Processing

This Section discusses the processing that occurs for the colormap notification event Colormap-Notify. The X server can report ColormapNotify events to clients wanting information about when the colormap changes and when a colormap is installed or uninstalled. The X server generates this event type whenever a client application:

- Changes the colormap member of the XSetWindowAttributes structure by calling XChangeWindowAttributes or XFreeColormap.
- Installs or uninstalls the colormap by calling XInstallColormap or XUninstallColormap.

To receive this event type in a client application, you pass the window ID of the window and ColormapChangeMask to the event_mask argument of XSelectInput.

The members of the XColormapEvent structure associated with this event are window, colormap, new, and state. The window member is set to the window ID of the window whose associated colormap is changed, installed, or uninstalled. For a colormap that is changed by a call to XChangeWindowAttributes, installed, or uninstalled, the colormap member is set to the colormap resource ID of the colormap associated with the window. For a colormap that is changed by a call to XFreeColormap, the colormap member is set to None. The new member is set to indicate whether the colormap for the specified window was changed or installed or uninstalled, and it can be either True or False. If True, the colormap was changed. If False, the colormap was installed or uninstalled. The state member is always set to indicate whether the colormap is installed or uninstalled, and it can be one of the corresponding constants ColormapInstalled or ColormapUninstalled.

8.4.9. Client Communication Event Processing

The following Sections discuss the processing that occurs for these client communication events: ClientMessage, PropertyNotify, SelectionClear, SelectionNotify, and SelectionRequest.

8.4.9.1. ClientMessage Event Processing

The X server generates ClientMessage events only when a client calls the function XSendEvent. See Section 8.8 in this Chapter for information on how to send an event.

The members of the XClientMessageEvent structure associated with this event are window, message_type, format, and data. The window member is set to the window ID of the window to which the event was sent. The message_type member is set to an atom that indicates how the data is to be interpreted by the receiving client. The format member is set to 8, 16, or 32 and specifies whether the data should be viewed as a list of bytes, shorts, or longs. The data member is a union that contains the members b, s, and l. The b, s, and l members represent data of twenty 8-bit values, ten 16-bit values, and five 32-bit values. Particular message types might not make use of all these values. The X server places no interpretation on the values in the message_type or data members.

8.4.9.2. PropertyNotify Event Processing

The X server can report PropertyNotify events to clients wanting information about property changes for a specified window. A property consists of an atom name, an atom type, a data format, and some property data. Generated with state PropertyNewValue when a property of the window is changed using XChangeProperty or XRotateProperties (even when adding zero-length data using XChangeProperty) and when replacing all or part of a property with identical data using XChangeProperty or XRotateProperties. Generated with state PropertyDeleted when a property of the window is deleted using XDeleteProperty or XGetProperty, if the delete argument is True.

See Chapter 4 for more information.

To receive this event type, you pass the window ID and PropertyChangeMask as the event_mask argument to XSelectInput.

The members of the XPropertyEvent structure associated with this event are window, atom, time, and state. The window member is set to the window ID of the window whose associated property was changed. The atom member is set to the property's atom and indicates which property was changed or desired. The time member is set to the server time when the property was changed. The state member is set to indicate whether the property was changed to a new value or deleted, and it can be one of the corresponding constants PropertyNewValue or PropertyDelete.

8.4.9.3. SelectionClear Event Processing

The X server reports SelectionClear events to the current owner of a selection. The X server generates this event type on the window losing ownership of the selection to a new owner. This sequence of events could occur whenever a client calls XSetSelectionOwner. See Section 4.4 for information on this function and on selections.

The members of the XSelectionClearEvent structure associated with this event are window, selection, and time. The window member is set to the ID of the window losing ownership of the selection. The selection member is set to the selection atom. The time member is set to the last change time recorded for the selection. The owner member is the window that was specified by the current owner in its XSetSelectionOwner call.

8.4.9.4. SelectionRequest Event Processing

The X server reports SelectionRequest events to the owner of a selection. The X server generates this event whenever a client requests a selection conversion by calling XConvertSelection, and the specified selection is owned by a window.

The members of the XSelectionRequestEvent structure associated with this event are owner, requestor, selection, target, property, and time. The owner member is set to the window ID of the window owning the selection. The owner member is the window that was specified by the current owner in its XSetSelectionOwner call. The requestor member is set to the window ID of the window requesting the selection. The selection member is set to the atom that indicates which selection. For example, PRIMARY is used to indicate the primary selection. The target member is set to the atom that indicates the type the selection is desired in. The property member can be the atom or the property or None. The time member is set to the time and is a timestamp, expressed in milliseconds, or CurrentTime from the ConvertSelection request.

The client whose window owns the selection should do the following:

- Convert the selection based on the atom contained in the target member.
- If a property was specified (that is, the property member is set), the client should store the result as that property on the requestor window and then send a SelectionNotify event to the requestor by calling XSendEvent with an empty event-mask, that is the event should be sent to the creator of the requestor window.
- If the selection cannot be converted as requested, the client should send a SelectionNotify event with the property set to None.

8.4.9.5. SelectionNotify Event Processing

This event is generated by the X server in response to a XConvertSelection request When there is an owner, it should be generated by the owner of the selection by using XSendEvent. The owner of a selection should send this event to a requestor when a selection has been converted and stored as a property. or when a selection conversion could not be performed (indicated with the property member set to None).

If None is specified as the property, the owner should choose a property name, store the result as that property on the requestor window, and then send a SelectionNotify to give that actual property name.

The members of the XSelectionEvent structure associated with this event are requestor, selection, target, property, and time. The requestor member is set to the window ID of the window associated with the requestor of the selection. The selection member is set to the atom that indicates the kind of selection. For example, PRIMARY is used for the primary selection. The target member is set to the atom that indicates the desired type. For example, PIXMAP is used for a pixmap. The property member is set to the atom that indicates which property the result was stored on. If none were possible, then the property member will be set to None. The time member is set to the time in which the conversion took place and can be a timestamp, expressed in milliseconds, or CurrentTime.

8.5. Selecting Events

There are two ways to select the events you want reported to your client application. One way is to set the event_mask member of the XSetWindowAttributes structure when you call XCreateWindow and XChangeWindowAttributes. See Chapter 3 for a discussion of these functions. Another way is to use XSelectInput. Section 8.4 discussed the processing that occurs for each event type associated with the event mask you pass to XSelectInput.

```
XSelectInput(display, w, event_mask)
    Display *display;
    Window w;
    unsigned long event_mask;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. Client applications interested in an event for a particular window pass that window's ID.
event_mask	Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.

The XSelectInput function requests that the X server report the events associated with the event masks that you pass to the event_mask argument. Initially, X will not report any of these events.

See Section 8.3 for a discussion of these event masks.

Events are reported relative to a window. If a window is not interested in an event, it usually propagates to the closest ancestor that is interested, unless the do_not_propagate mask prohibits it.

A call to XSelectInput overrides any previous call to XSelectInput for the same window from the same client but not for other clients. Multiple clients can select for the same events on the same window with the following restrictions:

- Multiple clients can select events on the same window because their event masks are disjoint. After the X server generates an event, it reports it to all interested clients.
- Only one client at a time can select CirculateRequest, ConfigureRequest, or MapRequest events, which are associated with the event mask SubstructureRedirectMask.
- Only one client at a time can select a ResizeRequest event, which is associated with the event mask ResizeRedirectMask.
- Only one client at a time can select a ButtonPress event, which is associated with the event mask ButtonPressMask.

The server reports the event to to all interested clients.

If a client passes both ButtonPressMask and ButtonReleaseMask for a specified window, a ButtonPress event in that window will automatically grab the mouse until all buttons are released and ButtonRelease events are sent to windows as described for XGrabPointer. This ensures that a window will see the ButtonRelease event corresponding to the ButtonPress event, even though the mouse may have exited the window in the meantime.

If a client passes PointerMotionMask, the X server sends MotionNotify events independent of the state of the pointer buttons. If, instead, the client passes one or more of Button1MotionMask, Button2MotionMask, Button3MotionMask, Button4MotionMask, Button5MotionMask, the X server generates MotionNotify events only when one or more of the specified buttons is depressed. These are used to request MotionNotify events only when particular buttons are held down.

XSelectInput can generate BadValue and BadWindow errors.

8.6. Handling the Output Buffer

Under some circumstances, many event functions discuss in the remainder of this Chapter also flush the output buffer.

The output buffer is an area used by the Xlib library to store requests. The functions described in this Section can flush the input queue if the function would block or not return an event. That is, all requests residing in the output buffer that have not yet been sent are transmitted to the X server. Conversely, these functions differ in the additional tasks they might perform. For example, XSync not only flushes the output buffer, but it can also discard all events on the event queue. The following paragraphs describe the XFlush and XSync functions.

To flush the output buffer, use XFlush.

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

display Specifies the connection to the X server.

The XFlush function flushes the output buffer. Most client applications need not use this function because the output buffer is automatically flushed the next time an event or reply is read from the output buffer by a call to XPending, XNextEvent, or XWindowEvent.

To flush the output buffer and then wait until all requests have been processed, use XSync.

XSync(*display*, *discard*)
    Display **display*;
    int *discard*;

display Specifies the connection to the X server.

discard Specifies whether XSync discards all events on the event queue. You can pass the value 0 or 1.

The XSync function flushes the output buffer and then waits until all requests have been received and processed by the X server. Any errors generated must be handled by the error handler. For each error event received and processed by the X server, XSync calls the client application's XError routine. Any events generated by the server are enqued into the library's event queue.

Finally, if you passed the value 0 to the discard argument, XSync does not discard the events on the queue. If you passed the value 1, this function discards all events on the queue, including those events that were on the queue before XSync was called. Client applications seldom need to call XSync.

8.7. Event Queue Management

Xlib maintains an event queue. However, the operating system may be buffering data in its network connection that is not yet read into the event queue. The following paragraphs describe the XEventsQueued and XPending functions.

To check the number of events in the event queue, use XEventsQueued.

int XEventQueued(*display*, *more*)
    Display **display*;
    int *mode*;

display Specifies the connection to the X server.

mode Specifies the mode. You can specify one of these constants: QueuedAlready, QueuedAfterFlush, QueuedAfterReading.

If mode is QueuedAlready, XEventsQueued returns the number of events already in the event queue (and never performs a system call). If mode is QueuedAfterFlush, it returns the number of events already in the queue, if it is nonzero. If there are no events in the queue, it flushes the output buffer, attempts to read more events out of the application's connection, and returns the number read. If mode is QueuedAfterReading, it returns the number of events already in the queue, if it is nonzero. If there are no events in the queue, it attempts to read more events out of the application's connection without flushing the output buffer and returns the number read.

XEventsQueued always returns immediately without I/O if there are events already in the queue. XEventsQueued with mode QueuedAfterFlush is identical in behavior to XPending. XEventsQueued with mode QueuedAlready is identical to the XQLength function (see Chapter 2).

To return the number of events that are pending, use XPending.

int XPending(*display*)
    Display **display*;

display Specifies the connection to the X server.

The XPending function returns the number of events that have been received from the X server but have not been removed from the event queue. (You remove events from the queue by calling XNextEvent or XWindowEvent.)

XPending is identical to XEventsQueued with the mode QueuedAfterFlush.

8.8. Manipulating the Event Queue

Xlib provides functions that let you manipulate the event queue. The next three Sections discuss how to:

- Obtain events in order and remove them from the queue.
- Peek at events in the queue without removing them.
- Obtain events that match the event mask or the arbitrary functions you provide.

8.8.1. Returning the Next Event

To get the next event and remove it from the queue, use XNextEvent.

```
XNextEvent(display, event_return)
    Display *display;
    XEvent *event_return;
```

display	Specifies the connection to the X server.
event_return	The XNextEvent function copies the event's associated structure into this client-supplied structure.

The XNextEvent function copies the first event from the event queue into the specified XEvent structure and then removes it from the queue. If the event queue is empty, XNextEvent flushes the output buffer and blocks until an event is received. For example, if a CreateNotify event is at the head of the queue, this function removes it and then copies the structure XCreateWindowEvent into XEvent.

To peek at the event queue, use XPeekEvent.

```
XPeekEvent(display, event_return)
    Display *display;
    XEvent *event_return;
```

display	Specifies the connection to the X server.
event_return	The XPeekEvent function copies the event's associated structure into this client-supplied structure.

The XPeekEvent function returns the first event from the event queue, but it does not remove the event from the queue. If the queue is empty, XPeekEvent flushes the output buffer and blocks until an event is received. It then copies the event into the client-supplied XEvent structure without removing it from the event queue. For example, if a CreateNotify event is at the head of the queue, this function peeks at it but does not remove it and then copies the structure XCreateWindowEvent into XEvent. You can use the QLength macro to determine if there are any events to peek at. For further information, see Chapter 2.

8.8.2. Selecting Events Using a Predicate Procedure

Each of the functions discussed in this Section requires you to pass a predicate procedure that determines if the event matches the one specified in the corresponding function. Your predicate procedure must decide only if the event is useful and must not call Xlib functions. In particular, it is called from inside the event routine, which must be locked so that the event queue is consistent in a multi-threaded environment.

The predicate procedure and its associated arguments are:

```
Bool (*predicate)(display, event, args)
   Display *display;
   XEvent *event;
   char *args;
```

display	Specifies the connection to the X server.
event	Specifies a pointer to the structure XEvent. This structure is a union of the individual structures declared for each event type.
args	Specifies the arguments passed in from the XIfEvent, XCheckIfEvent, or XPeekIfEvent function.

The predicate procedure is called once for each event in the queue until it finds a match between the event in the queue and the event specified by the corresponding function. After finding a match, the predicate procedure must return True or False if it did not find a match.

To check the event queue for the specified event and, if the events match, remove the event from the queue, use XIfEvent.

```
XIfEvent(display, event_return, predicate, arg)
    Display *display;
    XEvent *event_return;
    Bool (*predicate)();
    char *arg;
```

display	Specifies the connection to the X server.
event_return	The XIfEvent function copies the matched event's associated structure into this client-supplied structure.
predicate	Specifies the procedure that is to be called to determine if the next event in the queue matches the one specified by the event argument.
arg	Specifies the user-supplied argument that will be passed to the predicate procedure.

The XIfEvent function completes only when the specified predicate procedure returns a nonzero (true) for an event, which indicates an event on the queue matches the specified event. This predicate procedure is also called each time an event is added to the queue. XIfEvent flushes the output buffer if it blocks waiting for an event. XIfEvent removes the event from the queue and, when it returns, copies the structure into the client-supplied XEvent structure.

To check the event queue for the specified event without blocking, use XCheckIfEvent.

```
Bool XCheckIfEvent(display, event_return, predicate, arg)
    Display *display;
    XEvent *event_return;
    Bool (*predicate)();
    char *arg;
```

display	Specifies the connection to the X server.
event_return	The XCheckIfEvent function copies the matched event's associated structure into this client-supplied structure.
predicate	Specifies the procedure that is to be called to determine if the next event in the queue matches the one specified by the event argument.
arg	Specifies the user-supplied argument that will be passed to the predicate procedure.

When the predicate procedure finds a match, XCheckIfEvent copies the matched event into the client-supplied XEvent structure and returns True. (This event is removed from the queue.) If the predicate procedure finds no match, XCheckIfEvent returns False and flushes the output buffer. All earlier events stored in the queue are not discarded.

To check the event queue for the specified event but not remove the event from the queue, use XPeekIfEvent.

```
XPeekIfEvent(display, event_return, predicate, arg)
    Display *display;
    XEvent *event_return;
    Bool (*predicate)();
    char *arg;
```

display	Specifies the connection to the X server.
event_return	The XPeekIfEvent function copies the matched event's associated structure into this client-supplied structure.
predicate	Specifies the procedure that is to be called to determine if the next event in the queue matches the one specified by the event argument.
arg	Specifies the user-supplied argument that will be passed to the predicate procedure.

The XPeekIfEvent function returns only when the specified predicate procedure returns a nonzero (true) for the next event in the queue that matches the specified event. This predicate procedure is called each time an event is added to the queue. After the predicate procedure finds a match, XPeekIfEvent copies the matched event into the client-supplied XEvent structure without removing the event from the queue. XPeekIfEvent flushes the output buffer if it blocks waiting for an event.

8.8.3. Selecting Events Using a Window or Event Mask

These functions let you to select events by window or event types, which allows you to process events out of order. See Section 8.3. for a discussion of the valid event mask names.

To remove the next event that matches both a window and an event mask, use XWindowEvent.

```
XWindowEvent(display, w, event_mask, event_return)
    Display *display;
    Window w;
    long event_mask;
    XEvent *event_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose next matched event you want to remove.
event_mask	Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.
event_return	The XWindowEvent function copies the matched event's associated structure into this client-supplied structure.

The XWindowEvent function searches the event queue for an event that matches both the specified window and event mask. When it finds a match, XWindowEvent removes that event from the queue and copies it into the specified XEvent structure. The other events stored in the queue are not discarded. If the event you requested is not in the queue, XWindowEvent flushes the output buffer and blocks until one is received.

To remove the next event that matches both the passed window and the passed mask, use XCheckWindowEvent. This function is similar to XWindowEvent, except it never blocks, and it returns a Boolean indicating if the event was returned.

```
Bool XCheckWindowEvent(display, w, event_mask, event_return)
    Display *display;
    Window w;
    int event_mask;
    XEvent *event_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose next matched event you want to remove.
event_mask	Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.
event_return	The XCheckWindowEvent function copies the matched event's associated structure into this client-supplied structure.

The XCheckWindowEvent function searches the event queue, then the events available on the server connection, for the first event that matches the specified window and event mask. When it finds a match, XCheckWindowEvent removes that event, copies it into the specified XEvent structure, and returns True. The other events stored in the queue are not discarded. If the event you requested is not available, XCheckWindowEvent flushes the output buffer and returns False.

To remove the next event that matches an event mask, use XMaskEvent.

```
XMaskEvent(display, event_mask, event_return)
    Display *display;
    unsigned long event_mask;
    XEvent *event_return;
```

display	Specifies the connection to the X server.
event_mask	Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.
event_return	The XMaskEvent function copies the matched event's associated structure into this client-supplied structure.

The XMaskEvent function searches the event queue for the events associated with the specified mask. When it finds a match, XMaskEvent removes that and copies it into the specified XEvent structure. The other events stored in the queue are not discarded. If the event you requested is not in the queue, XMaskEvent flushes the output buffer and blocks until one is received.

To remove the next event, if any, that matches an event mask, use XCheckMaskEvent. This function is similar to XMaskEvent, except that it never blocks, and it returns a Boolean indicating if the event was returned.

```
Bool XCheckMaskEvent(display, event_mask, event_return)
    Display *display;
    unsigned long event_mask;
    XEvent *event_return;
```

display	Specifies the connection to the X server.
event_mask	Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.
event_return	The XCheckMaskEvent function copies the matched event's associated structure into this client-supplied structure.

The XCheckMaskEvent function searches first the event queue, then any events available on the server connection, for the first event that matches the specified mask. When it finds a match, XCheckMaskEvent removes that event, copies it into the specified XEvent structure, and returns True. The other events stored in the queue are not discarded. If the event you requested is not available, XCheckMaskEvent flushes the output buffer and returns False.

To return the next event in the queue that matches an event type, use XCheckTypedEvent.

Bool XCheckTypedEvent(*display*, *event_type*, *event_return*)
    Display **display*;
    int *event_type*;
    XEvent **event_return*;

display	Specifies the connection to the X server.
event_type	Specifies the event type to be compared.
event_return	The XCheckTypedEvent function copies the matched event's associated structure into this client-supplied structure.

The XCheckTypedEvent function searches first the event queue, then any events available on the server connection, for the first event that matches the specified type. When it finds a match, XCheckTypedEvent returns its associated event structure to the specified XEvent structure and returns True. The other events in the queue are not discarded. If the event is not available, XCheckTypedEvent flushes the output buffer and returns False.

To return the next matched event in the queue for the specified window, use XCheckTypedWindowEvent.

Bool XCheckTypedWindowEvent(*display*, *w*, *event_type*, *event_return*)
    Display **display*;
    Window *w*;
    int *event_type*;
    XEvent **event_return*;

display	Specifies the connection to the X server.
w	Specifies the window ID.
event_type	Specifies the event type to be compared.
event_return	The XCheckTypedWindowEvent function copies the matched event's associated structure into this client-supplied structure.

The XCheckTypedWindowEvent function searches first the event queue, then any events available on the server connection, for the first event that matches the specified type and window. When it finds a match, XCheckTypedWindowEvent removes the event from the queue, copies it into the specified XEvent structure and returns True. The other events in the queue are not discarded. If the event is not available, XCheckTypedWindowEvent flushes the output buffer and returns False.

8.9. Putting an Event Back onto the Queue

To push an event back to the top of the event queue, use XPutBackEvent.

XPutBackEvent(*display*, *event*)
    Display **display*;
    XEvent **event*;

display	Specifies the connection to the X server.
event	Specifies a pointer to the XEvent structure. This structure is a union of the individual structures declared for each event type.

The XPutBackEvent function pushes an event back onto the head of the display's event queue. This can be useful if you have read an event and then decide that you would rather deal with it later. There is no limit to how many times in succession you can call XPutBackEvent.

8.10. Sending Events to Other Applications

To send an event to a specified window, use XSendEvent. This function is often used in selection processing. For example, the owner of a selection should use XSendEvent to send a SelectionNotify event to a requestor when a selection has been converted and stored as a property.

Status XSendEvent(*display*, *w*, *propagate*, *event_mask*, *event_send*)
    Display **display*;
    Window *w*;
    Bool *propagate*;
    unsigned long *event_mask*;
    XEvent **event_send*;

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window interested in the event, and is referred to as the destination window. You can pass the window ID or either PointerWindow or InputFocus.
propagate	Specifies a boolean value that is either True or False.
event_mask	Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.
event_send	Specifies a pointer to the event that is to be sent.

The XSendEvent function identifies the destination window, determines which clients should receive the specified events, and ignores any active grabs. This function requires you to pass an event mask. See Section 8.3 for a discussion of the valid event mask names. This function uses the w argument to identify the destination window as follows:

- If you pass PointerWindow to w, the destination window is the window that contains the pointer.
- If you pass InputFocus to w, and if the focus window contains the pointer, the destination window is the window that contains the pointer. If the focus window does not contain the pointer, the destination window is the focus window.

To determine which clients should receive the specified events, XSendEvent uses the propagate argument as follows:

- If propagate is False, the event is sent to every client selecting on destination any of the event types in the event_mask argument.
- If propagate is True, and no clients have selected on destination any of the event types in event-mask, the destination is replaced with the closest ancestor of destination for which some client has selected a type in event-mask and for which no intervening window has that type in its do_not_propagate_mask. If no such window exists or if the window is an ancestor of the focus window and InputFocus was originally specified as the destination, the event is not sent to any clients. Otherwise, the event is reported to every client selecting on the final destination any of the types specified in event_mask.

The events in the XEvent structure must be one of the core events or one of the events defined by a loaded extension, so that the X server can correctly byte swap the contents as necessary. The contents of the event are otherwise unaltered and unchecked by the X server except to force send_event to True in the forwarded event and to set the sequence number in the event correctly.

XSendEvent returns zero if the conversion-to-wire protocol failed, otherwise, nonzero.

XSendEvent can generate BadValue and BadWindow errors.

8.11. Getting Pointer Motion History

Many X server implementations will maintain a more precise history of pointer motion between event notification. The pointer position at each pointer hardware interrupt may be stored into a buffer for later retrieval. This is called the motion history buffer. For example, a few applications, best exemplified by paint programs, want to have a precise history of where the pointer traveled. However, this is highly excessive for most applications.

To get the motion history for a specified window and time, use XGetMotionEvents.

```
XTimeCoord *XGetMotionEvents(display, w, start, stop, nevents_return)
    Display *display;
    Window w;
    Time start, stop;
    int *nevents_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose associated pointer motion events you want to retrieve.
start *stop*	Specify the time interval in which the events are returned from the motion history buffer. You can pass a time stamp, expressed in milliseconds, or CurrentTime. If the stop time is in the future, it is equivalent to specifying CurrentTime.
nevents_return	Returns the number of events from the motion history buffer.

The XGetMotionEvents function returns all events in the motion history buffer that fall between the specified start and stop times inclusive and that have coordinates that lie within (including borders) the specified window at its present placement. If the start time is later than the stop time or if the start time is in the future, no events are returned. The return type for this function is a structure defined as follows:

```
typedef struct {
        Time time;
        unsigned short x, y;
} XTimeCoord;
```

The time member is set to the time, in milliseconds. The x and y members are set to the coordinates of the pointer and are reported relative to the origin of the specified window. You should use XFree to free the data returned from this call. (See Section 2.4 for further information.)

XGetMotionEvents can generate a BadWindow error.

8.12. Handling Error Events

Xlib provides functions with which you can:

- Enable or disable synchronization
- Use the default error handlers

The following Sections discuss the functions associated with these tasks.

8.12.1. Enabling or Disabling Synchronization

When debugging X applications, it can be very convenient to require the library to behave synchronously, so that errors are reported at the time they occur. The routine below allows you to disable or enable synchronous behavior. Note that graphics may occur dramatically slower when enabled (30 or more times slower). On UNIX-based systems, there is also a global variable

_Xdebug that if set to nonzero before starting a program under a debugger will force synchronous library behavior.

To enable or disable synchronization, use XSynchronize.

```
int (*XSynchronize(display, onoff))()
    Display *display;
    int onoff;
```

display	Specifies the connection to the X server.
onoff	Specifies whether to enable or disable synchronization. Possible values you can pass are 0 (disable synchronization) or nonzero (enable synchronization).

The XSynchronize function returns the previous after function. If onoff is nonzero, XSynchronize turns on synchronous behavior. A value of zero resets the state to off (disable synchronous behavior).

After completing their work, all Xlib functions that generate protocol requests call what is known as a previous after function. XSetAfterFunction sets which function is to be called.

```
int (*XSetAfterFunction(display, proc))()
    Display *display;
    int (*proc)();
```

display	Specifies the connection to the X server.
proc	Specifies the function to be called after an Xlib function that generates a protocol request completes its work.

The specified proc will be called with only a display pointer. It returns the previous after function.

8.12.2. Using the Default Error Handlers

There are two default error handlers in the library, one to handle typically fatal conditions (for example, the connection to a display server dying due to a machine crash), and one to handle error events from the X server. These error handlers can be changed to user-supplied routines if you prefer your own error handling and can be changed as often as you like. If either of these routines are passed a NULL pointer, it will reinvoke the default handler. The default action of the supplied routine is to print an explanatory message and exit. This Section discusses: XSetErrorHandler, XGetErrorText, XGetErrorDatabaseText, XDisplayName, and XSetIOErrorHandler.

To handle error events, use XSetErrorHandler.

```
XSetErrorHandler(handler)
    int (*handler)(Display *, XErrorEvent *)
```

handler	Specifies the program's supplied error handler.

The program's supplied error hander generally is called by Xlib whenever an XError event is received. It is not called on BadName errors from OpenFont, LookupColor, AllocNamedColor, protocol requests, on BadFont errors from a QueryFont protocol request, or on BadAlloc or BadAccess errors. These errors generally are reflected back to the program through the procedureal interface. This is not assumed to be a fatal condition. It is acceptable for this procedure to return. However, the error handler should not perform any operations (directly or indirectly) on the display. The fields of the XErrorEvent structure passed to XError should be interpreted as follows:

```
typedef struct {
        int type
        Display *display;             /* Display the event was read from */
        unsigned long serial;          /* serial number of failed request */
        char error_code;               /* error code of failed request */
        char request_code;             /* Major op-code of failed request */
        char minor_code;               /* Minor op-code of failed request */
        XID resourceid;                /* resource id */
} XErrorEvent;
```

The serial member is the number of requests starting from one sent over the network connection since it was opened. It is the number that was the value of the request sequence number immediately after the failing call was made. The request_code member is a protocol representation of the name of the procedure that failed and are defined in <X11/X.h>. The following error codes can be returned by the functions described in this Chapter:

Error Code	Description
BadAccess	A client attempted to grab a key/button combination already grabbed by another client. A client attempted to free a colormap entry that it did not already allocate. A client attempted to store into a read-only colormap entry. A client attempted to modify the access control list from other than the local (or otherwise authorized) host. A client attempted to select an event type that another client has already selected, and, that at most, one client can select at a time.
BadAlloc	The server failed to allocate the requested resource or server memory.
BadAtom	A value for an Atom argument does not name a defined Atom.
BadColor	A value for a Colormap argument does not name a defined Colormap.
BadCursor	A value for a Cursor argument does not name a defined Cursor.
BadDrawable	A value for a Drawable argument does not name a defined Window or Pixmap.
BadFont	A value for a Font or GContext argument does not name a defined Font.
BadGC	A value for a GContext argument does not name a defined GContext.
BadIDChoice	The value chosen for a resource identifier is either not included in the range assigned to the client or is already in use. Under normal circumstances this cannot occur and should be considered a server or X Library error.
BadImplementation	The server does not implement some aspect of the request. A server that generates this error for a core request is deficient. As such, this error is not listed for any of the requests. However, clients should be prepared to receive such errors and either handle or discard them.

Error Code	Description
BadLength	The length of a request is shorter or longer than that minimally required to contain the arguments. This usually means an internal Xlib error.
BadMatch	In a graphics request, the root and depth of the GC does not match that of the drawable.
	An InputOnly window is used as a Drawable.
	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
	An InputOnly window locks this attribute.
	The values do not exist for an InputOnly window.
BadName	A font or color of the specified name does not exist.
BadPixmap	A value for a Pixmap argument does not name a defined Pixmap.
BadRequest	The major or minor opcode does not specifies a valid request.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a window argument does not name a defined window.

Note

The BadAtom, BadColor, BadCursor, BadDrawable, BadFont, BadGC, BadPixmap, and BadWindow errors are also used when the argument type is extended by union with a set of fixed alternatives (for example, Window, PointerRoot, or None.)

To obtain textual descriptions of the specified error code, use XGetErrorText.

XGetErrorText(*display*, *code*, *buffer_return*, *length*)
    Display **display*;
    int *code*;
    char **buffer_return*;
    int *length*;

display	Specifies the connection to the X server.
code	Specifies the error code for which you want to obtain a description.
buffer_return	Returns the error description.
length	Specifies the size of the buffer.

The XGetErrorText function copies a null-terminated string describing the specified error code into the specified buffer. It is recommended that you use this routine to obtain an error description because extensions to the Xlib library may define their own error codes and error strings.

To obtain error messages from the error database, use XGetErrorDatabaseText.

```
XGetErrorDatabaseText(display, name, message, default_string, buffer_return, length)
    Display display;
    char *name, *message;
    char *default_string;
    char *buffer_return;
    int length;
```

display	Specifies the connection to the X server.
name	Specifies the name of the application.
message	Specifies the type of the error message.
default_string	Specifies the default error message.
buffer_return	Returns the error description.
length	Specifies the size of the buffer.

The XGetErrorDatabaseText function returns a message (or the default message) from the error message database. Given a pair of strings as keys, XGetErrorDatabaseText uses the resource manager to look up a string and returns 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 /usr/lib/XerrorDB.

The name argument should generally be the name of your application. The message argument should indicate which type of error message you want. Three predefined sets of names are used by Xlib to report errors:

XProtoError	The protocol error number is used as a string for the message argument.
XlibMessage	These are the message strings that are used internally by the library.
XRequestMajor	The major request protocol number is used for the message argument. If no string is found in the error database, the default_string is returned to the buffer argument.

To report an error to the user when the requested display does not exist, use XDisplayName.

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

string	Specifies the character string.

The XDisplayName function returns the name of the display that you are currently using. If a NULL string is specified, XDisplayName looks in the environment for the display and returns the display name that the user was requesting. This makes it easier to report to the user precisely which display the program attempted to open when the initial connection attempt failed.

To handle fatal I/O errors, use XSetIOErrorHandler.

```
XSetIOErrorHandler(handler)
    int (* handler)(Display *);
```

handler	Specifies the program's supplied error handler.

The XSetIOErrorHandler sets the fatal IO error handler. The program's supplied error handler will be called by Xlib if any sort of system call error occurs. For example, the connection to the server was lost. This is assumed to be a fatal condition. That is, the called routine should not return. If the IO error handler does return, the client process will exit.

Chapter 9

Predefined Property Functions

There are a number of predefined properties for common information usually associated with windows. The atoms for these predefined properties can be found in <X11/Xatom.h>, where the prefix ''XA_'' is added to each atom name.

Xlib provides functions with which you can perform predefined property operations. This Chapter discusses how to:

- Communicate with window managers
- Manipulate standard colormaps

9.1. Communicating with Window Managers

This Section discusses a set of properties and functions that are necessary for clients to communicate effectively with window managers. Some of these properties have complex structures. Because all the data in a single property on the server has to be of the same format (8, 16, or 32 bit), and because the C structures representing property types cannot be guaranteed to be uniform in the same way, Set and Get functions are provided for properties with complex structures to format.

These functions define but do not enforce minimal policy among window managers. Writers of window managers are urged to use the information in these properties rather than invent their own properties and types. A window manager writer, however, can define additional properties beyond this proposed least common denominator.

In addition to Set and Get functions for individual properties, Xlib includes one function, XSetStandardProperties, that sets all or portions of several properties. The purpose of XSetStandardProperties is to provide a simple interface for the programmer who wants to code an application in an afternoon. Applications which need to communicate to the window manager more information than is possible with XSetStandardProperties should not use this interface. Instead, they should call the Set functions for the additional or specific properties that they need.

In order to work well with most window managers, every application should specifies the following information:

- Name of the application
- Name string for the icon
- Command used to invoke the application
- Size and window manager hints as defined below

Xlib does not set defaults for the properties described in this Section. Thus, the default behavior is determined by the window manager and may be based on the presence or absence of certain properties. All the properties are considered to be hints to a window manager. When implementing window management policy, a window manager determines what to do with this information and can ignore it.

The supplied properties are:

Name	Type	Format	Description
WM_NAME	STRING	8	Name of the application
WM_ICON_NAME	STRING	8	Name to be used in icon
WM_NORMAL_HINTS	WM_SIZE_HINTS	32	Size hints for a window in its normal state. The C type of this property is XSizeHints (see below).
WM_ZOOM_HINTS	WM_SIZE_HINTS	32	Size hints for a zoomed window. The C type of this property is XSizeHints (see below).
WM_HINTS	WM_HINTS	32	Additional hints set by client for use by the window manager. The C type of this property is XWMHints (see below).
WM_COMMAND	STRING	8	The command and arguments, separated by ASCII 0s, used to invoke the application.
WM_ICON_SIZE	WM_ICON_SIZE	32	The window manager may set this property on the root window to specifies the icon sizes it supports. The C type of this property is XIconSize (see below).
WM_CLASS	STRING	32	Set by application programs to allow window and session managers to obtain the application's resources from the resource database.
WM_TRANSIENT_FOR	WINDOW	32	Set by application programs to indicate to the window manager that a transient top-level window, such as a dialog box, is not really a full-fledged window.

The atom names stored in <X11/Xatom.h> are named "XA_*PROPERTY_NAME*".

Xlib provides functions with which you can set and get predefined properties. Note that calling the Set function for a property with complex structure redefines all fields in that property, even though only some of those fields may have a specified new value. Simple properties for which Xlib does not provide a Set or Get function may be set using XChangeProperty and their values may be retrieved using XGetWindowProperty. The remainder of this Section discusses how to:

- Set standard properties
- Set and get the name of a window
- Set and get the icon name of a window
- Set the command atom
- Set and get window manager hints
- Set and get window sizing hints

- Set and get icon size hints
- Set and get the class of a window
- Set and get the transient property for a window

9.1.1. Setting Standard Properties

To specifies a minimum set of properties describing the "quickie" application, use XSetStandardProperties. This function sets all or portions of the WM_NAME, WM_ICON_NAME, WM_HINTS, WM_COMMAND, WM_NORMAL_HINTS properties.

```
XSetStandardProperties(display, w, window_name, icon_name, icon_pixmap, argv, argc, hints)
    Display *display;
    Window w;
    char *window_name;
    char *icon_name;
    Pixmap icon_pixmap;
    char **argv;
    int argc;
    XSizeHints *hints;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
window_name	Specifies the name of the window.
icon_name	Specifies the name to be displayed in the window's icon.
icon_pixmap	Specifies the single plane pixmap that is to be used for the icon or None.
argv	Specifies a pointer to the command and arguments used to start the application.
argc	Specifies the number of arguments.
hints	Specifies a pointer to the sizing hints for the window in its normal state.

The XSetStandardProperties function provides a means by which, with a single call, simple applications set the most essential properties. XSetStandardProperties should be used to give a window manager some information about your program's preferences. It should not be used by applications which need to communicate more information than is possible with XSetStandardProperties. See Section 9.1.6 for a discussion of the XSizeHints structure.

XSetStandardProperties can generate BadAlloc and BadWindow errors.

9.1.2. Setting and Getting Window Names

Xlib provides functions to set and read the name of a window. These functions set and read the WM_NAME property.

To assign a name to a window, use XStoreName.

```
XStoreName(display, w, window_name)
    Display *display;
    Window w;
    char *window_name;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window to which you want to assign a name.
window_name	Specifies the name of the window. The name should be a null-terminated string. This name will be returned by any subsequent call to XFetchName.

The XStoreName function assigns the name passed to window_name to the specified window. A window manager may display the window name in some prominent place, such as the title bar, to allow users to identify windows easily. Some window managers may display a window's name in the window's icon, although they are encouraged to use the window's icon name if one is provided by the application.

XStoreName can generate BadAlloc and BadWindow errors.

To get the name of a window, use XFetchName.

```
int XFetchName(display, w, window_name_return)
    Display *display;
    Window w;
    char **window_name_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose name you want a pointer set to.
window_name_return	Returns a pointer to the window name, which will be a null-terminated string.

The XFetchName function obtains a window name and returns either a nonzero if it succeeds or zero if it fails (for example, if no name has been set for the argument window). If the WM_NAME property has not been set for this window, XFetchName sets this argument to NULL. When finished with it, a client must free the name string using the free library subroutine.

XFetchName can generate a BadWindow error.

9.1.3. Setting and Getting Icon Names

Xlib provides functions to set and read the name to be displayed in a window's icon. These functions set and read the WM_ICON_NAME property.

To set the name to be displayed in a window's icon, use XSetIconName.

```
XSetIconName(display, w, icon_name)
    Display *display;
    Window w;
    char *icon_name;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose icon name is being set.
icon_name	Specifies the name to be displayed in the window's icon. The name should be a null-terminated string. This name is returned by any subsequent call to XGetIconName.

XSetIconName can generate BadAlloc and BadWindow errors.

To get the name a window wants displayed in its icon, use XGetIconName.

```
int XGetIconName(display, w, icon_name_return)
    Display *display;
    Window w;
    char **icon_name_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID. This is the window whose icon name you want a pointer set to.

icon_name_return
Returns a pointer to the name to be displayed in the window's icon. The name will be a null-terminated string.

The XGetIconName function obtains the window name to be displayed in its icon and either returns a nonzero if it succeeds or zero if it fails (for example, if no icon name has been set for the argument window). If you never assigned a name to the window, XGetIconName sets this argument to NULL. When finished with it, a client must free the icon name string using the free library subroutine.

XGetIconName can generate a BadWindow error.

9.1.4. Setting the Command Atom

To set the value of the command atom, use XSetCommand. This function sets the WM_COMMAND property.

```
XSetCommand(display, w, argv, argc)
    Display *display;
    Window w;
    char **argv;
    int argc;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
argv	Specifies a pointer to the command and arguments used to start the application.
argc	Specifies the number of arguments.

The XSetCommand function records the command and arguments used to invoke the application.

XSetCommand can generate BadAlloc and BadWindow errors.

9.1.5. Setting and Getting Window Manager Hints

The functions discussed in this Section set and read the WM_HINTS property and use the XWMHints structure, as defined in the <X11/Xutil.h> header file:

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

The definitions for the flags field are:

```
#define InputHint                  (1L << 0)
#define StateHint                  (1L << 1)
#define IconPixmapHint             (1L << 2)
#define IconWindowHint             (1L << 3)
#define IconPositionHint           (1L << 4)
#define IconMaskHint               (1L << 5)
#define WindowGroupHint            (1L << 6)
#define AllHints (InputHint|StateHint|IconPixmapHint|IconWindowHint| \
```

IconPositionHint|IconMaskHint|WindowGroupHint)

The input field is used to communicate to the window manager the input focus model used by the application. Applications which expect input but never explicitly set focus to any of their subwindows (use the push model of focus management), such as X10-style applications that use real-estate driven focus, should set this field to True. Similarly, applications that set input focus to their subwindows only when it is given to their top-level window by a window manager should also set this field to True. Applications which manage their own input focus by explicitly setting focus to one of their subwindows whenever they want keyboard input (that is, use the pull model of focus management) should set this field to False. Applications which never expect any keyboard input should also set this field to False.

Pull model window managers should make it possible for push model applications to get input by setting input focus to the top level windows of applications whose input field is True. Push model window managers should make sure that pull model applications do not break them, by resetting input focus to PointerRoot when appropriate (for example, whenever an application whose input field is False sets input focus to one of its subwindows).

The definitions for the initial_state flag are:

```
#define DontCareState     0   /* don't know or care */
#define NormalState       1   /* most applications want to start this way */
#define ZoomState         2   /* application wants to start zoomed */
#define IconicState       3   /* application wants to start as an icon */
#define InactiveState     4   /* application believes it is seldom used;
                                 some wm's may put it on inactive menu */
```

The icon_mask specifies which pixels of the icon_pixmap should be used as the icon. Both the icon_pixmap and icon_mask must be single plane pixmaps. The icon_window lets an application provide a window for use as an icon for window managers that support such use. This allows for nonrectangular pixmaps. The window_group lets you specifies that this window belongs to a group of other windows. For example, if a single application manipulates multiple children of the root window, this allows you to provide enough information that a window manager can iconify all of the windows, rather than just the one window.

To set the value of the window manager hints atom, use XSetWMHints.

XSetWMHints(*display*, *w*, *wmhints*)
    Display **display*;
    Window *w*;
    XWMHints **wmhints*;

display	Specifies the connection to the X server.
w	Specifies the window ID.
wmhints	Specifies a pointer to the window manager hints.

The XSetWMHints function sets the window manager hints that include icon information and location, the initial state of the window, and whether the application relies on the window manager to get keyboard input.

XSetWMHints can generate BadAlloc and BadWindow errors.

To read the value of the window manager hints atom, use XGetWMHints.

XWMHints *XGetWMHints(*display*, *w*)
    Display **display*;
    Window *w*;

display	Specifies the connection to the X server.
w	Specifies the window ID.

The XGetWMHints function reads the value of the window manager hints atom and returns either NULL if it fails (for example, if no WM_HINTS property was set on window w) or a pointer to a XWMHints structure if it succeeds. When finished with this function, an application must free the space used for that structure by calling XFree. (See Section 2.4 for further information).

XGetWMHints can generate a BadWindow error.

9.1.6. Setting and Getting Window Sizing Hints

Xlib provides functions with which you can set or get window sizing hints.

The functions discussed in this Section use the XSizeHints structure, as defined in the <X11/Xutil.h> header file:

```
typedef struct {
        long flags;                        /* marks which fields in this 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;
```

The definitions for the flags field are:

```
#define USPosition          (1L << 0)   /* user specified x, y */
#define USSize              (1L << 1)   /* user specified width, height */
#define PPosition           (1L << 2)   /* program specified position */
#define PSize               (1L << 3)   /* program specified size */
#define PMinSize            (1L << 4)   /* program specified minimum size */
#define PMaxSize            (1L << 5)   /* program specified maximum size */
#define PResizeInc          (1L << 6)   /* program specified resize increments */
#define PAspect             (1L << 7)   /* program specified min and max aspect ratios */
#define PAllHints (PPosition|PSize|PMinSize|PMaxSize|PResizeInc|PAspect)
```

The x, y, width, and height elements describe a desired position and size for the window. To indicate that this information was specified by the user, set the USPosition and USSize flags. To indicate that it was specified by the application without any user involvement, set PPosition and PSize. This allows a window manager to know that the user specifically asked where the window should be placed or how the window should be sized and that the window manager does not have to rely on the program's opinion.

The min_width and min_height elements specifies the minimum size that the window can be for the application to be useful. The max_width and max_height elements specifies the maximum size. The width_inc and height_inc elements define an arithmetic progression of sizes (minimum to maximum) into which the window prefers to be resized. The min_aspect and max_aspect elements are expressed as ratios of x and y, and they allow an application to specifies the range of aspect ratios it prefers.

The next two functions set and read the WM_NORMAL_HINTS property.

To set the size hints for a window in its normal state, use XSetNormalHints.

```
void XSetNormalHints(display, w, hints)
    Display *display;
    Window w;
    XSizeHints *hints;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
hints	Specifies a pointer to the sizing hints for the window in its normal state.

The XSetNormalHints function sets the size hints structure for the specified window. Applications use XSetNormalHints to inform the window manager of the size or position desirable for that window. In addition, an application that wants to move or resize itself should call XSetNormalHints and specifies its new desired location and size, instead of making direct X calls to move or resize. This is because window managers may ignore redirected configure requests, but they pay attention to property changes.

To set size hints, an application not only must assign values to the appropriate elements in the hints structure, but it also must set the flags field of the structure to indicate which information is present and where it came from. A call to XSetNormalHints is meaningless, unless the flags field is set to indicate which elements of the structure have been assigned values.

XSetNormalHints can generate BadAlloc and BadWindow errors.

To return the size hints for a window in its normal state, use XGetNormalHints.

```
Status XGetNormalHints(display, w, hints_return)
    Display *display;
    Window w;
    XSizeHints *hints_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
hints_return	Returns the sizing hints for the window in its normal state.

The XGetNormalHints function returns in its last argument the size hints for a window in its normal state. It returns either a nonzero status if it succeeds or zero if it fails (for example, the application specified no normal size hints for this window).

XGetNormalHints can generate a BadWindow error.

The next two functions set and read the WM_ZOOM_HINTS property.

To set the value of the zoom hints atom, use XSetZoomHints.

```
XSetZoomHints(display, w, zhints)
    Display *display;
    Window w;
    XSizeHints *zhints;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
zhints	Specifies a pointer to the zoom hints.

Many window managers think of windows in three states: iconic, normal, or zoomed. The XSetZoomHints function provides the window manager with information for the window in the zoomed state.

XSetZoomHints can generate BadAlloc and BadWindow errors.

To read the value of the zoom hints atom, use XGetZoomHints.

Status XGetZoomHints(*display*, *w*, *zhints_return*)
    Display **display*;
    Window *w*;
    XSizeHints **zhints_return*;

display	Specifies the connection to the X server.
w	Specifies the window ID.
zhints_return	Returns the zoom hints.

The XGetZoomHints function returns in its last argument the size hints for a window in its zoomed state. It returns either a nonzero status if it succeeds or zero if it fails (for example, the application specified no zoom size hints for this window).

XGetZoomHints can generate a BadWindow error.

To set the value of any property of type WM_SIZE_HINTS, use XSetSizeHints.

XSetSizeHints(*display*, *w*, *hints*, *property*)
    Display **display*;
    Window *w*;
    XSizeHints **hints*;
    Atom *property*;

display	Specifies the connection to the X server.
w	Specifies the window ID.
hints	Specifies a pointer to the size hints.
property	Specifies the property atom.

The XSetSizeHints function sets the XSizeHints structure for the named property and the specified window. This is used by XSetNormalHints and XSetZoomHints, and can be used to set the value of any property of type WM_SIZE_HINTS. Thus, it may be useful if other properties of that type get defined.

XSetSizeHints can generate BadAlloc, BadAtom, and BadWindow errors.

To read the value of any property of type WM_SIZE_HINTS, use XGetSizeHints.

Status XGetSizeHints(*display*, *w*, *hints_return*, *property*)
    Display **display*;
    Window *w*;
    XSizeHints **hints_return*;
    Atom *property*;

display	Specifies the connection to the X server.
w	Specifies the window ID.
hints_return	Returns the size hints.
property	Specifies the property atom.

XGetSizeHints returns the XSizeHints structure for the named property and the specified window. This is used by XGetNormalHints and XGetZoomHints. It also can be used to retrieve the value of any property of type SIZE_HINTS. Thus, it may be useful if other properties of that type get defined. XGetSizeHints returns a nonzero status if a size hint was defined and zero otherwise.

XGetSizeHints can generate BadAtom and BadWindow errors.

9.1.7. Setting and Getting Icon Sizing Hints

Applications can cooperate with window managers by providing icons in sizes supported by a window manager. To communicate the supported icon sizes to the applications, a window manager should set the icon size property on the root window. To find out what icon sizes a window manager supports, applications should read the icon size property from the root window.

The functions discussed in this Section set or read the WM_ICON_SIZE property. In addition, they use the XIconSize structure, as defined in the <X11/Xutil.h> header file:

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

The width_inc and height_inc elements define an arithmetic progression of sizes (minimum to maximum) that represent the supported icon sizes.

To set the value of the icon size atom, use XSetIconSizes.

```
XSetIconSizes(display, w, size_list, count)
    Display *display;
    Window w;
    XIconSize *size_list;
    int count;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
size_list	Specifies a pointer to the size list.
count	Specifies the number of items in the size list.

The XSetIconSizes function is used only by window managers to set the supported icon sizes.

XSetIconSizes can generate BadAlloc and BadWindow errors.

To return the value of the icon sizes atom, use XGetIconSizes.

```
Status XGetIconSizes(display, w, size_list_return, count_return)
    Display *display;
    Window w;
    XIconSize **size_list_return;
    int *count_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
size_list_return	Returns a pointer to the size list.
count_return	Returns the number of items in the size list.

The XGetIconSizes function returns zero if a window manager has not set icon sizes and nonzero status otherwise. This function should be called by an application which wants to find out what icon sizes would be most appreciated by the window manager under which the application is running. The application should then use XSetWMHints to supply the window manager with an icon pixmap or window in one of the supported sizes.

XGetIconSizes can generate a BadWindow error.

9.1.8. Setting and Getting the Class of a Window

Xlib provides functions to set and get the class of a window. These functions set and read the WM_CLASS property. In addition, they use the XClass structure, as defined in the <X11/Xutil.h> header file:

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

The res_name member contains the application name, while the res_class member contains the application class. Note that the name set in this property may differ from the name set as WM_NAME. That is, WM_NAME specifies what should be displayed in the title bar and, therefore, may contain temporal information (for example, the name of a file currently in an editor's buffer). On the other hand, the name specified as part of WM_CLASS is the formal name of the application that should be used when retrieving the application's resources from the resource database.

To set the class of a window, use XSetClassHint.

XSetClassHint(*display*, *w*, *class_hints*)
  Display **display*;
  Window *w*;
  XClassHint **class_hints*;

display	Specifies the connection to the X server.
w	Specifies the window ID.
class_hints	Specifies a pointer to a XClassHint structure that is to be used.

The XSetClassHint functions sets the class hint for the specified window.

XSetClassHint can generate BadAlloc and BadWindow errors.

To get the class of a window, use XGetClassHint.

Status XGetClassHint(*display*, *w*, *class_hints_return*)
  Display **display*;
  Window *w*;
  XClassHint **class_hints_return*;

display	Specifies the connection to the X server.
w	Specifies the window ID.
class_hints_return	Returns the XClassHints structure.

The XGetClassHint function obtains the class of the specified window.

XGetClassHint can generate a BadWindow error.

9.1.9. Setting and Getting the Transient Property

An application may want to indicate to the window manager that a transient top-level window (for example, a dialog box) is not really a full-fledged window. Rather, it is operating on behalf of (or is transient for) another window. To do so, the application would set the WM_TRANSIENT_FOR property of the dialog box to be the window handle of its main window. Some window managers may use this information to unmap an application's dialog boxes (for example, when the main application window gets iconified).

The functions discussed in this Section set and read the WM_TRANSIENT_FOR property.

To set the WM_TRANSIENT_FOR property for a window, use XSetTransientForHint.

```
XSetTransientForHint(display, w, prop_window)
    Display *display;
    Window w;
    Window prop_window;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
prop_window	Specifies the window ID that the WM_TRANSIENT_FOR property is to be set to.

The XSetTransientForHint set the WM_TRANSIENT_FOR atom of the specified window to the specified prop_window.

XSetTransientForHint can generate BadAlloc and BadWindow errors.

To get the WM_TRANSIENT_FOR value for a window, use XGetTransientForHint.

```
Status XGetTransientForHint(display, w, prop_window_return)
    Display *display;
    Window w;
    Window *prop_window_return;
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
prop_window_return	Returns the WM_TRANSIENT_FOR property of the specified window.

The XGetTransientForHint function obtains the WM_TRANSIENT_FOR property for the specified window.

XGetTransientForHint can generate a BadWindow error.

9.2. Manipulating Standard Colormaps

Applications with color palettes, smooth-shaded drawings, or digitized images demand large numbers of colors. In addition, these application often require an efficient mapping from color triples to pixel values that display the appropriate colors.

As an example, consider a 3D display program that wants to draw a smoothly shaded sphere. At each pixel in the image of the sphere, the program computes the intensity and color of light reflected back to the viewer. The result of each computation is a triple of red, green, and blue coefficients in the range 0.0 to 1.0. To draw the sphere, the program needs a colormap that provides a large range of uniformly distributed colors. The colormap should be arranged so that the program can convert its RGB triples into pixel values very quickly, because drawing the entire sphere will require many such conversions.

On many current workstations the display is limited to 256 or fewer colors. Applications must allocate colors carefully not only to make sure they cover the entire range they need but also to make use of as many of the available colors as possible. On a typical X display, many applications are active at once. Most workstations have only one hardware lookup table for colors, so only one application colormap can be installed at a given time. The application using the installed colormap is displayed correctly, while the other applications ''go technicolor'' and are displayed with false colors.

As another example, consider a user who is running an image processing program to display earth-resources data. The image processing program needs a colormap set up with 8 reds, 8 greens, and 4 blues (total of 256 colors). Because some colors are already in use in the default colormap, the image processing program allocates and installs a new colormap.

The user decides to alter some of the colors in the image. He invokes a color palette program to mix and choose colors. The color palette program also needs a colormap with 8 reds, 8 greens, and 4 blues, so, just as the image-processing program, it must allocate and install a new colormap.

Because only one colormap can be installed at a time, the color palette may be displayed incorrectly whenever the image-processing program is active. Conversely, whenever the palette program is active, the image may be displayed incorrectly. The user can never match or compare colors in the palette and image. Contention for colormap resources can be reduced if applications with similar color needs share colormaps.

As another example, the image processing program and the color palette program could share the same colormap if there existed a convention that described how the colormap was set up. Whenever either program was active, both would be displayed correctly.

The standard colormap properties define a set of commonly used colormaps. Applications that share these colormaps and conventions display true colors more often and provide a better interface to the user.

9.2.1. Standard Colormaps

Standard colormaps allow applications to share commonly used color resources. This allows many applications to be displayed in true colors simultaneously, even when each application needs an entirely filled colormap.

Several standard colormaps are described. Usually, these colormaps are created by a window manager. Applications should use the standard colormaps if they already exist. If the standard colormaps do not exist, applications should create new standard colormaps.

The XStandardColormap structure contains:

```
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;
} XStandardColormap;
```

The colormap field is the ID of a colormap created by the XCreateColormap function. The red_max, green_max, and blue_max fields give the maximum red, green, and blue values, respectively. Each color coefficient ranges from zero to its max, inclusive. For example, a common colormap allocation is 3/3/2 (3 planes for red, 3 planes for green, and 2 planes for blue). This colormap would have red_max = 7, green_max = 7, and blue_max = 3. An alternate allocation that uses only 216 colors is red_max = 5, green_max = 5, and blue_max = 5.

The red_mult, green_mult, and blue_mult fields give the scale factors used to compose a full pixel value. (See the discussion of the base_pixel field for further information.) For a 3/3/2 allocation red_mult might be 32, green_mult might be 4, and blue_mult might be 1. For a 6-colors-each allocation, red_mult might be 36, green_mult might be 6, and blue_mult might be 1.

The base_pixel field gives the base pixel value used to compose a full pixel value. Usually, the base_pixel is obtained from a call to the XAllocColorPlanes function. Given integer red, green, and blue coefficients in their appropriate ranges, one then can compute a corresponding pixel value by using the following expression:

```
r * red_mult + g * green_mult + b * blue_mult + base_pixel
```

For gray-scale colormaps, only the colormap, red_max, red_mult, and base_pixel fields are defined. The other fields are ignored.

To compute a gray-scale pixel value, use the following expression.

```
gray * red_mult + base_pixel
```

The properties containing the XStandardColormap information have the type RGB_COLOR_MAP.

9.2.2. Standard Colormap Properties and Atoms

Several standard colormaps are available. Each standard colormap is defined by a property, and each such property is identified by an atom. The following list names the atoms and describes the colormap associated with each one. The <X11/Xatom.h> header file contains the definitions for each of the following atoms, which are prefixed with "XA_".

RGB_DEFAULT_MAP

This atom names a property. The value of the property is an XStandardColormap.

The property defines an RGB subset of the system default colormap. Some applications only need a few RGB colors and may be able to allocate them from the system default colormap. This is the ideal situation, since the fewer colormaps are active in the system the more applications are displayed with correct colors at all times.

A typical allocation for the RGB_DEFAULT_MAP on 8-plane displays is 6 reds, 6 greens, and 6 blues. This gives 216 uniformly distributed colors (6 intensities of 36 different hues) and still leaves 40 elements of a 256-element colormap available for special-purpose colors for text, borders, and so on.

RGB_BEST_MAP

This atom names a property. The value of the property is an XStandardColormap.

The property defines the best RGB colormap available on the display. (Of course, this is a subjective evaluation.) Many image processing and 3D applications need to use all available colormap cells and distribute as many perceptually distinct colors as possible over those cells. This implies that there may be more green values available than red, as well as more green or red than blue.

On an 8-plane pseudocolor display, RGB_BEST_MAP is a 3/3/2 allocation. On a 24-plane directcolor display, RGB_BEST_MAP is an 8/8/8 allocation. On other displays, RGB_BEST_MAP is purely up to the implementor of the display.

RGB_RED_MAP
RGB_GREEN_MAP
RGB_BLUE_MAP

These atoms name properties. The values of the properties are XStandardColormaps.

The properties define all-red, all-green, and all-blue colormaps, respectively. These maps are used by applications that want to make color-separated images. For example, a user might generate a full-color image on an 8-plane display both by rendering an image three times (once with high color resolution in red, once with green, and once with blue) and by multiply-exposing a single frame in a camera.

RGB_GRAY_MAP

This atom names a property. The value of the property is an XStandardColormap.

The property describes the best gray-scale colormap available on the display. As previously mentioned, only the colormap, red_max, red_mult, and base_pixel fields of the

XStandardColormap structure are used for gray-scale colormaps.

9.2.3. Getting and Setting the XStandardColormap Structure

To get the XStandardColormap structure associated with one of the described atoms, use XGetStandardColormap.

```
Status XGetStandardColormap(display, w, cmap_return, property)
    Display *display;
    Window w;
    XStandardColormap *cmap_return;
    Atom property;                          /* RGB_BEST_MAP, etc. */
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
cmap_return	Returns the color map associated with the specified atom.
property	Specifies the property atom.

The XGetStandardColormap function returns the colormap definition associated with the atom supplied as the property argument. For example, to fetch the standard gray-scale colormap for a display, you use XGetStandardColormap with the following syntax:

```
XGetStandardColormap(dpy, DefaultRootWindow(dpy), &cmap, RGB_GRAY_MAP);
```

Once you have fetched a standard colormap, you can use it to convert RGB values into pixel values. For example, given an XStandardColormap structure 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 can generate BadAtom and BadWindow errors.

To create or change a standard colormap, use XSetStandardColormap.

```
void XSetStandardColormap(display, w, cmap, property)
    Display *display;
    Window w;
    XStandardColormap *cmap;
    Atom property;                          /* RGB_BEST_MAP, etc. */
```

display	Specifies the connection to the X server.
w	Specifies the window ID.
cmap	Specifies the color map ID.
property	Specifies the property atom.

The XSetStandardColormap function usually is only used by window managers. To create a standard colormap, follow this procedure:

1. Grab the server.
2. See if the property is on the property list of the root window for the display.
3. If the desired property is not present, do the following:
   - Create a colormap (not required for RGB_DEFAULT_MAP)

- Determine the color capabilities of the display.
- Call XAllocColorPlanes or XAllocColorCells to allocate cells in the colormap.
- Call XStoreColors to store appropriate color values in the colormap.
- Fill in the descriptive fields in the property.
- Attach the property to the root window.

4. Ungrab the server.

XSetStandardColormap can generate BadAlloc, BadAtom, and BadWindow errors.

Chapter 10

Application Utility Functions

Once you have initialized the X system, you can use the Xlib utility functions to:

- Handle keyboard events
- Obtain the X environment defaults
- Parse the window geometry
- Parse the hardware colors
- Generate regions
- Manipulate regions
- Use the cut and paste buffers
- Determine the appropriate visual
- Manipulate images
- Manipulate bitmaps
- Use the resource manager
- Use the context manager

Note

As a group, the functions discussed in this Chapter provide the functionality that is frequently needed and spans toolkits. Many of these functions do not generate actual protocol requests to the server.

10.1. Keyboard Utility Functions

This Section discusses:

- Keyboard event functions
- Keysym classification macros

10.1.1. Keyboard Event Functions

The X server does not predefine the keyboard to be ASCII characters. It is often useful to know that the ''a'' key was just pressed or possibly that it was just released. When a key is pressed or released, the X server sends keyboard events to client programs. The structures associated with keyboard events contain a keycode member that assigns a number to each physical key on the keyboard. See Section 8.4.1 for a discussion of keyboard event processing. See Section 7.9 for information on how to manipulate the keyboard encoding.

Because keycodes are completely arbitrary and may differ from server to server, client programs wanting to deal with ASCII text, for example, must explicitly convert the keycode value into ASCII. The transformation of keycodes to ASCII or other character sets is arbitrary. Therefore, Xlib provides functions to help you customize the keyboard layout. Keyboards often differ dramatically, so writing code that presumes the existence of a particular key on the main keyboard will create portability problems. It may also be difficult to receive KeyRelease events on certain X server implementations because of hardware or software restrictions.

Keyboard events are usually sent to the smallest enclosing window which is interested in that type of event underneath the pointer's position. It is also possible to assign the keyboard input focus to a specific window. When the input focus is attached to a window, keyboard events will go to the client which has selected input on that window rather than the window under the pointer.

This Section discusses the functions with which you can query the keyboard, look up the keyboard mappings, rebind the keyboard, or use an alternate keyboard mapping file.

Warning

> Some implementations cannot support KeyRelease events. You should think seriously before designing software that takes advantage of KeyRelease events if you are concerned about wide portability, though there are some applications that can exploit KeyRelease events to provide superior user interfaces. You should also be very careful when selecting which keys may be used in such applications. It may be impossible to guarantee the existence of a set of keys on all keyboards with the probable exception of a-z, spacebar, and carriage return.

To look up the KeySyms, use XLookupKeysym.

KeySym XLookupKeysym(*event_key*, *index*)
    XKeyEvent **event_key*;
    int *index*;

event_key	Specifies the key event that is to be used. This event is either a KeyPress event or a KeyRelease event.
index	Specifies the index into the KeySyms table.

The XLookupKeysym function uses a given keyboard event and the index you specified to return the KeySym from the list that corresponds to the keycode member in the XKeyPressedEvent or XKeyReleasedEvent structure.

To refresh the stored modifier and keymap information, use XRefreshKeyboardMapping.

XRefreshKeyboardMapping(*event_map*)
    XMappingEvent **event_map*;

event_map	Specifies the mapping event that is to be used.

The XRefreshKeyboardMapping function refreshes the stored modifier and keymap information. You usually call this function when a MappingNotify event occurs. The result is to update a client application's knowledge of the keyboard. See Section 8.4.6.7 for information on MappingNotify event processing.

To map a key event to an ASCII string, use XLookupString.

int XLookupString(*event_struct*, *buffer_return*, *bytes_buffer*, *keysym_return*, *status_return*)
    XKeyEvent **event_struct*;
    char **buffer_return*;
    int *bytes_buffer*;
    KeySym **keysym_return*;
    XComposeStatus **status_return*;

event_struct	Specifies the key event structure to be used: XKeyPressedEvent or XKeyReleasedEvent.
buffer_return	Returns the translated characters.
bytes_buffer	Specifies the length of the buffer. No more than bytes_buffer of translation are returned.
keysym_return	If this argument is not NULL, returns the keysym computed from the event.

status_return	Specifies either a pointer to the XCompose structure that is to contain compose key state information and that allows compose key processing to take place, or NULL.

The XLookupString function is a convenience routine that can be used to map a key event to an ASCII string, using the modifier bits in the key event to deal with shift, lock, and control. It returns the translated string into the user's buffer. It also detects any rebound keysyms (see XRebindKeysym) and returns the specified bytes. XLookupString returns, as its value, the length of the string stored in the tag buffer. If the lock modifier has a caps_lock key associated with it, XLookupString interprets the lock modifier to perform caps lock processing.

To rebind the meaning of a keysym for a client, use XRebindKeysym.

XRebindKeysym(*display, keysym, list, mod_count, string, bytes_string*)
    Display **display*;
    KeySym *keysym*;
    KeySym **list*;
    int *mod_count*;
    unsigned char **string*;
    int *bytes_string*;

display	Specifies the connection to the X server.
keysym	Specifies the keysym to be rebound.
list	Specifies a pointer to an array of keysyms that are being used as modifiers.
mod_count	Specifies the number of modifiers in the modifier list.
string	Specifies a pointer to the string that is to be returned by XLookupString.
bytes_string	Specifies the length of the string.

The XRebindKeysym function can be used to rebind the meaning of a keysym for the client. It does not redefine the keycode in the X server but merely provides an easy way for long strings to be attached to keys. XLookupString returns this string when the appropriate set of modifier keys are pressed and when the keysym would have been used for the translation. Note that you can rebind a keysym that may not exist.

To convert the name of the keysym to the keysym code, use XStringToKeysym.

KeySym XStringToKeysym(*string*)
    char **string*;

string	Specifies the name of the keysym that is to be converted.

Valid keysym names are listed in <X11/keysymdef.h>. If the specified string does not match a valid keysym, XStringToKeysym returns NoSymbol.

To convert a keysym code to the name of the keysym, use XKeysymToString.

char *XKeysymToString(*keysym_str*)
    KeySym *keysym_str*;

keysym_str	Specifies the keysym that is to be converted.

The returned string is in a static area and must not be modified. If the specified keysym is not defined, XKeysymToString returns a NULL.

To convert a key code to a defined keysym, use XKeycodeToKeysym.

```
KeySym XKeycodeToKeysym(display, keycode, index_return)
    Display *display;
    KeyCode keycode;
    int index_return;
```

display	Specifies the connection to the X server.
keycode	Specifies the keycode.
index_return	Returns the element of keycode vector.

XKeycodeToKeysym uses internal Xlib tables, which already have converted alphabetic upper-case to lowercase, and returns the keysym defined for the specified keycode and the element of the keycode vector.

To convert a keysym to the appropriate keycode, use XKeysymToKeycode.

```
KeyCode XKeysymToKeycode(display, keysym_kcode)
    Display *display;
    Keysym keysym_kcode;
```

display	Specifies the connection to the X server.
keysym_kcode	Specifies the keysym that is to be searched for.

If the specified keysym is not defined for any keycode, XKeysymToKeycode returns zero.

10.1.2. Keysym Classification Macros

You may want to test if a keysym of the defined set (XK_MISCELLANY) is, for example, on the key-pad or the function keys. You can use the keysym macros to perform the following tests.

IsKeypadKey(*keysym*)

Returns True if the keysym is on the keypad.

IsCursorKey(*keysym*)

Returns True if the keysym is on the cursor key.

IsPFKey(*keysym*)

Returns True if the keysym is on the PF keys.

IsFunctionKey(*keysym*)

Returns True if the keysym is on the function keys.

IsMiscFunctionKey(*keysym*)

Returns True if the keysym is on the miscellaneous function keys.

IsModifierKey(*keysym*)

Returns True if the keysym is on the modifier keys.

10.2. Obtaining the X Environment Defaults

A program often needs a variety of options in the X environment (for example, fonts, colors, mouse, background, text, cursor). Specifiesing these options on the command line is inefficient and unmanageable because individual users have a variety of tastes with regard to window appearance. XGetDefault makes it easy to find out the fonts, colors, and other environment defaults favored by a particular user. Defaults are usually loaded into the RESOURCE_MANAGER property on the root window at login. If no such property exists, a resource file in the user's home directory is loaded. On a 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 pathname 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. See Section 10.11 for details of the format of these files.

XGetDefaults provides a simple interface for clients not wishing to use the X Toolkit or the more elaborate interfaces provided by the resource manager discussed in Sxection 10.11.

The strings returned by XGetDefault are owned by Xlib and should not be modified or freed by the client. The definition for this function is:

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

display	Specifies the connection to the X server.
program	Specifies the program name for the Xlib defaults. You must pass the program name in with the program argument (usually argv[0]).
option	Specifies the option name.

XGetDefault returns the value NULL if the option name specified in this argument does not exist for the program.

10.3. Parsing the Window Geometry

Xlib provides functions with which you can parse the window geometry.

To parse standard window geometry strings, use XParseGeometry.

```
int XParseGeometry(parsestring, x_return, y_return, width_return, height_return)
    char *parsestring;
    int *x_return, *y_return;
    int *width_return, *height_return;
```

parsestring	Specifies the string you want to parse.
x_return *y_return*	Returns the xoffset and yoffset determined.
width_return *height_return*	Return the width and height determined.

By convention, X applications use a standard string to indicate window size and placement. XParseGeometry makes it easier to conform to this standard because it allows you to parse the standard window geometry. Specifically, this function lets you parse strings of the form:

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

The items in this form map into the arguments associated with this function.

The XParseGeometry function returns a bitmask that indicates which of the four values (width, height, xoffset, and yoffset) were actually found in the string and that indicates whether the x and y values are negative. By convention, -0 is not equal to +0, because the user needs to be able to say "position the window relative to the right or bottom edge." For each value found, the corresponding argument is updated. For each value not found, the argument is left unchanged. The bits are represented by these constants: XValue, YValue, WidthValue, HeightValue, XNegative, YNegative and are defined in <X11/Xutil.h>. They will be set whenever one of the values are defined or signs are set.

If the function returns either the XValue or YValue flag, you should place the window at the requested position. The border width (bwidth), size of the increments width and height (typically font width and height), and any additional interior space (xadder and yadder) are passed in to make it easy to compute the resulting size. It is not normally used by user programs, which typically use the XCreateWindow or XCreateSimpleWindow function to create the window.

To parse window geometry given an argument and a default position, use XGeometry.

```
int XGeometry(display, screen, position, default_position, bwidth, fwidth, fheight, xadder,
          yadder, x_return, y_return, width_return, height_return)
    Display *display;
    int screen;
    char *position, *default_position;
    unsigned int bwidth;
    unsigned int fwidth, fheight;
    int xadder, yadder;
    int *x_return, *y_return;
     int *width_return, *height_return;
```

display	Specifies the connection to the X server.
screen	Specifies the screen.
position	
default_position	Specify the geometry specifications.
bwidth	Specifies the border width.
fheight	
fwidth	Specify the font height and width in pixels (increment size).
xadder	
yadder	Specify additional interior padding needed in the window.
x_return	
y_return	Returns the xoffset and yoffset determined.
width_return	
height_return	Return the width and height determined.

The XGeometry function returns the position the window should be placed given a position and a default position. XGeometry determines the placement of a window using the current format to position windows. Given a fully qualified default geometry specification and, possibly, an incompletely specified geometry specification, it will return a bitmask value as defined above in the XParseGeometry call.

10.4. Parsing the Color Specifications

To parse color values, use XParseColor.

```
Status XParseColor(display, cmap, spec, exact_def_return)
      Display *display;
      Colormap cmap;
      char *spec;
      XColor *exact_def_return;
```

display	Specifies the connection to the X server.
cmap	Specifies the color map ID.
spec	Specifies the color name string. Uppercase and lowercase characters are acceptable.

*exact_def_return*Returns the exact colors for later use and sets the DoRed, DoGreen, and DoBlue flags.

The XParseColor function provides a simple way to create a standard user interface to color. It takes a string specification of a color, typically from a command line or XGetDefault option, and returns the corresponding red, green, and blue values that are suitable for a subsequent call to XAllocColor or XStoreColor. The color can be specified either as a color name (as in XAllocNamedColor) or as an initial sharp sign character following by a numeric specification, in one of the following formats:

#RGB	(4 bits each)
#RRGGBB	(8 bits each)
#RRRGGGBBB	(12 bits each)
#RRRRGGGGBBBB	(16 bits each)

The R, G, and B represent single hexadecimal digits (upper- or lower-case). When fewer than 16 bits each are specified, they represent the most significant bits of the value. For example, #3a7 is the same as #3000a0007000. The colormap is used only to determine which screen to look up the color on. For example, you can use the screen's default colormap.

This routine will fail and return zero status either if the initial character is a sharp sign, but the string otherwise fails to fit of the above formats, or if the initial character is not a sharp sign and the named color does not exist in the server's database.

XParseColor can generate a BadColor error.

10.5. Generating Regions

Regions are arbitrary collections of pixels. Xlib provides functions for manipulating regions. The opaque type Region is defined in <X11/Xutil.h>.

To generate a region from points, use XPolygonRegion.

```
Region XPolygonRegion(points, n, fill_rule)
    XPoint points[];
    int n;
    int fill_rule;
```

points	Specifies an array of points.
n	Specifies the number of points in the polygon.
fill_rule	Specifies the fill rule you want to set for the specified graphics context. You can pass one of these constants: EvenOddRule or WindingRule.

The XPolygonRegion function returns a region defined by the points array. See XCreateGC in Section 5.3 for an explanation of fill_rule.

To generate the smallest enclosing rectangle in rect, use XClipBox.

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

r	Specifies the region. This is the region in which the rectangle is located.
rect	Specifies the rectangle. This is the rectangle in which the smallest enclosing rectangle is generated.

The XClipBox function generates the smallest enclosing rectangle in the specified rectangle that is located in the specified region.

10.6. Manipulating Regions

Xlib provides functions with which you can manipulate regions. This Section discusses how to:

- Create, copy, or destroy regions
- Move or shrink regions
- Compute with regions
- Determine if regions are empty or equal
- Locate a point or rectangle in a region

10.6.1. Creating, Copying, or Destroying Regions

Xlib provides functions with which you can create, copy, or destroy a region.

To create a new empty region, use XCreateRegion.

```
Region XCreateRegion()
```

To set the graphics contexts to the specified region, use XSetRegion.

```
XSetRegion(display, gc, r)
    Display *display;
    GC gc;
    Region r;
```

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
r	Specifies the region. This is the region in which you want to set the specified graphics context.

The XSetRegion function sets the clip mask in the graphics contexts to the specified region. Once is is set in the GC, the region can be destroyed.

To deallocate the storage associated with a specified region, use XDestroyRegion.

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

r	Specifies the region.

10.6.2. Moving or Shrinking Regions

Xlib provides functions with which you can move or shrink regions.

To move the specified region by a specified amount, use XOffsetRegion.

```
XOffsetRegion(r, dx, dy)
    Region r;
    int dx, dy;
```

r	Specifies the region.
dx *dy*	Specify the x and y coordinates. These coordinates define the amount by which you want to move the specified region.

To reduce the specified region by a specified amount, use XShrinkRegion.

XShrinkRegion(*r*, *dx*, *dy*)
    Region *r*;
    int *dx*, *dy*;

r	Specifies the region.
dx *dy*	Specify the x and y coordinates. These coordinates define the amount by which you want to shrink the specified region.

Positive values shrink the size of the region, while negative values expand the region.

10.6.3. Computing with Regions

Xlib provides functions with which you can compute the interSection, union, or results of two regions.

To compute the interSection of two regions, use XIntersectRegion.

XIntersectRegion(*sra*, *srb*, *dr*)
    Region *sra*, *srb*, *dr*;

sra *srb*	Specify the two regions with which you want to perform the computation.
dr	Stores the result of the computation.

To compute the union of two regions, use XUnionRegion. The definition for this function is:

XUnionRegion(*sra*, *srb*, *dr*)
    Region *sra*, *srb*, *dr*;

sra *srb*	Specify the two regions with which you want to perform the computation.
dr	Stores the result of the computation.

To create a union of a source region and a rectangle, use XUnionRectWithRegion.

XUnionRectWithRegion(*rectangle*, *src_region*, *dest_region*)
    Rectangle **rectangle*;
    Region *src_region*;
    Region *dest_region*;

rectangle	Specifies the rectangle.
src_region	Specifies the source region to be used.
dest_region	Specifies the destination region.

The XUnionRectWithRegion function creates the destination region from a union of the specified rectangle and the specified source region.

To subtract two regions, use XSubtractRegion.

XSubtractRegion(*sra*, *srb*, *dr*)
    Region *sra*, *srb*, *dr*;

sra *srb*	Specify the two regions with which you want to perform the computation.
dr	Stores the result of the computation.

XSubtractRegion subtracts region B from region A.

To calculate the difference between the union and interSection of two regions, use XXorRegion.

```
XXorRegion(sra, srb, dr)
    Region sra, srb, dr;
```

sra
srb Specify the two regions with which you want to perform the computation.
dr Stores the result of the computation.

10.6.4. Determining If Regions Are Empty or Equal

Xlib provides functions with which you can determine if regions are empty or equal.

To determine if the specified region is empty, use XEmptyRegion.

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

r Specifies the region.

The XEmptyRegion function returns nonzero if the region is empty.

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

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

r1
r2 Specify the two regions. These are the regions you want to determine have the same offset, size, and shape.

The XEqualRegion function returns nonzero if the two regions are identical (that is, they have the same offset, size, and shape).

10.6.5. Locating a Point or a Rectangle in a Region

Xlib provides functions with which you can determine if a point or a rectangle is contained in a region.

To determine if a specified point resides in a specified region, use XPointInRegion.

```
int XPointInRegion(r, x, y)
    Region r;
    int x, y;
```

r Specifies the region.

x
y Specify the x and y coordinates. These define the coordinates of the point.

The XPointInRegion function returns nonzero if the point x, y is contained in the region r.

To determine if a specified rectangle resides in the specified region, use XRectInRegion. The definition for this function is:

```
int XRectInRegion(r, x, y, width, height)
    Region r;
    int x, y;
    unsigned int width, height;
```

r Specifies the region.

x

y	Specify the x and y coordinates. These define the coordinates of the point.
width *height*	Specify the width and height. These specifies the rectangle in which you want the point to reside.

The XRectInRegion function returns RectangleIn if the rectangle is entirely in the specified region, RectangleOut if the rectangle is entirely out of the specified region, and RectanglePart if the rectangle is partially in the specified region.

10.7. Using the Cut and Paste Buffers

Xlib provides functions with which you can use cut and paste buffers for programs using this form of communication. See also the Sections on selections, which are a more useful mechanism for interchanging data between clients, as typed information can be exchanged. X provides areas of memory in which bytes can be stored for implementing cut and paste between windows (implemented by use of properties on the first root window of the display). It is up to applications to agree on how to represent the data in the buffers. The data is most often ISO Latin 1 text. The atoms for eight such buffers are provided, which can be accessed as a ring or as explicit buffers (numbered 0 through 7). These buffers are typically used by either XYO applications or Andrew applications. We encourage new applications to share data by using selections. See Section 4.4 for further information.

To store data in cut buffer 0, use XStoreBytes.

```
XStoreBytes(display, bytes, nbytes)
    Display *display;
    char bytes[];
    int nbytes;
```

display	Specifies the connection to the X server.
bytes	Specifies the string of bytes you want stored. The byte string is not necessarily ASCII or null-terminated.
nbytes	Specifies the number of bytes of the bytes argument that you want stored.

The XStoreBytes function returns the number of bytes to be stored to the nbytes argument. Note that the cut buffer's contents need not be text, so null bytes are not special. The cut buffer's contents may be retrieved later by any client calling XFetchBytes.

XStoreBytes can generate BadAlloc and BadWindow errors.

To store data in a specified cut buffer, use XStoreBuffer.

```
XStoreBuffer(display, bytes, nbytes, buffer)
    Display *display;
    char bytes[];
    int nbytes;
    int buffer;
```

display	Specifies the connection to the X server.
bytes	Specifies the string of bytes you want stored. The byte string is not necessarily ASCII or null-terminated.
nbytes	Specifies the number of bytes of the bytes argument that you want stored.
buffer	Specifies the buffer in which you want to store the byte string.

XStoreBuffer can generate BadAlloc, BadAtom, and BadWindow errors.

To return data from cut buffer 0, use XFetchBytes.

```
char *XFetchBytes(display, nbytes_return)
    Display *display;
    int *nbytes_return;
```

display	Specifies the connection to the X server.
nbytes_return	Returns the number of bytes in the string in the buffer. XFetchBytes returns the number of bytes in this argument. If there is no data in the buffer, the value zero is returned to this argument.

The XFetchBytes function returns the number of bytes in the nbytes argument, if the buffer contains data. Otherwise, the function returns NULL and sets nbytes to 0. The XFetchBytes function returns the number of bytes in the nbytes argument, if the buffer contains data. Otherwise, the function returns NULL and sets nbytes to zero. The appropriate amount of storage is allocated and the pointer returned. The client must free this storage when finished with it by calling XFree. (See Section 2.4 for further information.) 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.

XFetchBytes can generate a BadWindow error.

To return data from a specified cut buffer, use XFetchBuffer.

```
char *XFetchBuffer(display, nbytes_return, return_buffer)
    Display *display;
    int *nbytes_return;
    int return_buffer;
```

display	Specifies the connection to the X server.
nbytes_return	Returns the number of bytes in the string in the buffer.
return_buffer	Specifies which buffer you want to stored data to be returned from.

The XFetchBuffer function returns the value zero to the nbytes argument if there is no data in the buffer.

XFetchBuffer can generate a BadValue error.

To rotate the cut buffers, use XRotateBuffers.

```
XRotateBuffers(display, rotate)
    Display *display;
    int rotate;
```

display	Specifies the connection to the X server.
rotate	Specifies how much to rotate the cut buffer.

The XRotateBuffers function rotates the cut buffers, such that buffer 0 becomes buffer n, buffer 1 becomes n+1 mod 8, and so on. This cut buffer numbering is global to the display. Note that XRotateBuffers will generate an error if any of the eight buffers have not been created.

XRotateBuffers can generate BadAtom, BadMatch, and BadWindow errors.

10.8. Determining the Appropriate Visual Type

A single display may support multiple screens. Each screen can have several different visual types supported at different depths. You can use the functions described in this Section to determine which visual to use for your application.

The functions in this Section use the XVisualInfo structure, which can be found in <X11/Xutil.h>.

```
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 definitions used for the visual information mask (vinfo_mask) 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
```

To obtain a list of visual information structures that match a specified template, use XGet-VisualInfo.

XVisualInfo *XGetVisualInfo(*display*, *vinfo_mask*, *vinfo_template*, *nitems_return*)
    Display **display*;
    long *vinfo_mask*;
    XVisualInfo **vinfo_template*;
    int **nitems_return*;

display	Specifies the connection to the X server.
vinfo_mask	Specifies the visual mask value.
vinfo_template	Specifies the visual attributes that are to be used in matching the visual structures.
nitems_return	Returns the number of matching visual structures.

The XGetVisualInfo function returns a list of visual structures that match the attributes specified by the template argument. If no visual structures match the template using the specified vinfo_mask, XGetVisualInfo returns a NULL. To free the data returned by this function, use XFree. (See Section 2.4 for further information.)

To obtain the visual information that matches the specified depth and class of the screen, use XMatchVisualInfo. The definition for this function is:

Status XMatchVisualInfo(*display*, *screen*, *depth*, *class*, *vinfo_return*)
    Display **display*;
    int *screen*;
    int *depth*;
    int *class*;
    XVisualInfo **vinfo_return*;

display	Specifies the connection to the X server.
screen	Specifies the screen.

depth	Specifies the depth of the screen.
class	Specifies the class of the screen.
vinfo_return	Returns the match visual information.

The XMatchVisualInfo function returns the visual information for a visual that matches the specified depth and class for a screen. Because multiple visuals that match the specified depth and class can exist, the exact visual chosen is undefined. If a visual is found, this function returns True, and the information on the visual is returned to the vinfo argument. Otherwise, when a visual is not found, it returns False.

10.9. Manipulating Images

Xlib provides several functions that perform basic operations on images. All operations on images are defined using an XImage structure, as defined in <X11/Xlib.h>. Because the number of different types of image formats can be very large, this hides details of image storage properly from applications.

This Section describes the functions for generic operations on images. Manufacturers can provide very fast implementations of these for the formats frequently encountered on their hardware. These functions are neither sufficient nor desirable to use for general image processing. Rather, they are here to provide minimal functions on screen format images. The basic operations for getting and putting images are XGetImage and XPutImage. See Chapter 6 for further information about these functions.

Note

The functions to read and write images to and from disk files are not, as yet, defined.

Most of the fields are defined in the core protocol to specifies hardware variants, bit and byte ordering you may encounter across manufacturers. See Section 6.7 for a description of the XImage structure.

The XImage structure describes an image as it exists in the client's memory. The user may request that some of the members such as height, width, and xoffset be changed when the image is sent to the server. Note that bytes_per_line in concert with offset can be used to extract the image to be only the subset. Other members (for example, byte order, bitmap_unit, and so forth) are characteristics of both the image and of the server. If these members differ between the image and the server, XPutImage makes the appropriate conversions. See Chapter 6 for information about XPutImage. If the image is formatted as an XYPixmap, that is the format member is set to the constant XYPixmap, the first byte of the first line of plane n must be located at the address (data + (n * height * bytes_per_line)).

To allocate sufficient memory for an XImage structure, use XCreateImage.

```
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;
```

display	Specifies the connection to the X server.
visual	Specifies a pointer to the visual.
depth	Specifies the depth of the image.
format	Specifies the format for the image. You can pass one of these constants: XYPixmap or ZPixmap.
offset	Specifies the number of pixels to ignore at the beginning of the scanline. This permits the rapid displaying of the image without requiring each scanline to be shifted into position.
data	Specifies a pointer to the image data.
width	Specifies the width (in pixels) of the image.
height	Specifies the height (in pixels) of the image.
bitmap_pad	Specifies the quantum of a scanline. In other words, the start of one scanline is separated in client memory from the start of the next scanline by an integer multiple of this many bits. You must pass one of these values: 8, 16, or 32.
bytes_per_line	Specifies the number of bytes in the client image between the start of one scanline and the start of the next. If you pass a zero value, Xlib assumes that the scanlines are contiguous in memory and calculates the value of bytes_per_line itself.

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

Note that when the image is created using XCreateImage or XGetImage, the destroy procedure that the XDestroyImage macro calls usually frees both the image structure and the data pointed to by the image structure.

The basic functions used to get a pixel, set a pixel, create a subimage, and add a constant offset to a Z format image are defined in the image object. The macros to call through the image object are defined in <X11/Xutil.h>.

To obtain a pixel value in an image, use XGetPixel.

```
unsigned long XGetPixel(ximage, x, y)
    XImage *ximage;
    int x;
    int y;
```

ximage	Specifies a pointer to the image.
x	
y	Specify the x and y coordinates.

The XGetPixel function returns the specified pixel from the named image. The X and Y coordinates are relative to the origin (upper-left [0,0]) of the image. The pixel value is returned in normalized format (that is, the least significant byte of the long is the least significant byte of the pixel).

To set a pixel value in an image, use XPutPixel.

```
int XPutPixel(ximage, x, y, pixel)
    XImage *ximage;
    int x;
    int y;
    unsigned long pixel;
```

ximage Specifies a pointer to the image.

x
y Specify the x and y coordinates.

pixel Specifies the new pixel value.

The XPutPixel function overwrites the pixel in the named image with the specified pixel value. The X and Y coordinates are relative to the origin (upper-left [0,0]) of the image. The input pixel value must be in normalized format. That is, the least significant byte of the long is the least significant byte of the pixel.

To create a subimage, use XSubImage.

```
XImage *XSubImage(ximage, x, y, subimage_width, subimage_height)
    XImage *ximage;
    int x;
    int y;
    unsigned int subimage_width;
    unsigned int subimage_height;
```

ximage Specifies a pointer to the image.

x
y Specify the x and y coordinates.

*subimage_width*Specifies the width (in pixels) of the new subimage.

*subimage_height*Specifies the height (in pixel) of the new subimage.

The XSubImage function creates a new image that is a subsection of an existing one. It allocates the memory.necessary for the new XImage structure and returns a pointer to the new image. The algorithm used is repetitive calls to XGetPixel and XPutPixel. Therefore, this function may be very slow.

To increment each pixel in the pixmap by a constant value, use XAddPixel.

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

ximage Specifies a pointer to the image.

value Specifies the constant value that is to be added.

The XAddPixel function adds a constant value to every pixel in an image. This function is very useful when you have a base pixel value from allocating color resources and need to manipulate the image to that form.

To deallocate the memory allocated in a previous call to XCreateImage, use XDestroyImage.

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

ximage Specifies a pointer to the image.

The XDestroyImage function deallocates the memory associated with the XImage structure.

Note that when the image is created using XCreateImage or XGetImage, the destroy procedure that this macro calls usually frees both the image structure and the data pointed to by the image structure.

10.10. Manipulating Bitmaps

Xlib provides functions with which you can read a bitmap from a file, save a bitmap to a file, or create a bitmap. This Section describes those functions that transfer bitmaps to and from the client's file system, thus allowing their reuse in a later connection (for example, from an entirely different client or to a different display or server).

To read a bitmap in from disk, use XReadBitmapFile.

```
int XReadBitmapFile(display, d, filename, width_return, height_return, bitmap_return, x_hot_return,
            y_hot_return)
    Display *display;
    Drawable d;
    char *filename;
    unsigned int *width_return, *height_return;
    Pixmap *bitmap_return;
    int *x_hot_return, *y_hot_return;
```

display	Specifies the connection to the X server.
d	Specifies the drawable.
filename	Specifies the file name to use. The format of the file name is operating system specific.
width_return *height_return*	Returns the width and height values of the read in bitmap file.
bitmap_return	Returns the bitmap ID that is created.
x_hot_return *y_hot_return*	Returns the hot spot coordinates.

The XReadBitmapFile function reads in a file containing a bitmap. The file can be either in the standard X version 10 format (that is, the format used by X version 10 bitmap program) or in the newer X version 11 bitmap format. If the file cannot be opened, XReadBitmapFile returns BitmapOpenFailed. If the file can be opened but does not contain valid bitmap data, it returns BitmapFileInvalid. If insufficient working storage is allocated, it returns BitmapNoMemory. If the file is readable and valid, it returns BitmapSuccess.

XReadBitmapFile assigns the bitmap's height and width, as read from the file, to the caller's variables width and height. It then creates a pixmap of the appropriate size, reads the bitmap data from the file into the pixmap and assigns the pixmap to the caller's variable bitmap. The caller must free the bitmap using XFreePixmap when done. If x_hot and y_hot are non-NULL, XReadBitmapFile sets *x_hot and *y_hot to the value of the hot spot as defined in the file. If no hot spot is defined, XReadBitmapFile sets *x_hot and *y_hot to -1,-1.

To write out a bitmap to a file, use XWriteBitmapFile.

```
int XWriteBitmapFile(display, filename, bitmap, width, height, x_hot, y_hot)
    Display *display;
    char *filename;
    Pixmap bitmap;
    unsigned int width, height;
    int x_hot, y_hot;
```

display	Specifies the connection to the X server.

filename Specifies the file name to use. The format of the file name is operating system specific.

bitmap Specifies the bitmap to be written.

width
height Specify the width and height. These are the dimensions of the bitmap to be written.

x_hot
y_hot Specifies where to place the hot spot coordinates (or -1,-1 if none are present) in the file.

The XWriteBitmapFile function writes a bitmap out to a file. The file is written out in X version 11 bitmap format, which is the format used by the X version 11 bitmap program. Refer to that program's man pages for details. While XReadBitmapFile can read in either X version 10 format or X version 11 format, XWriteBitmapFile always writes out X version 11 format only. If the file cannot be opened for writing, it returns BitmapOpenFailed. If insufficient memory is allocated it returns BitmapNoMemory. Otherwise, on no error, it returns BitmapSuccess. If x_hot and y_hot are not -1, -1, XWriteBitmapFile writes them out as the hot spot coordinates for the bitmap.

To create a pixmap and then perform a bitmap-format of the data into it, use XCreatePixmapFromBitmapData.

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

display Specifies the connection to the X server.

d Specifies the drawable.

data Specifies the data in bitmap format.

width
height Specify the width and height. These are the dimensions of the pixmap to create.

fg
bg Specifies the foreground and background pixel values to use.

depth Specifies the depth of the pixmap.

XCreatePixmapFromBitmapData creates a pixmap of the given depth and then does a bitmap-format XPutImage of the data into it. This is a convenience routine that clients could do manually.

To include a bitmap written out by XWriteBitmapFile in a program directly, as opposed to reading it in every time at run time, use XCreateBitmapFromData.

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

display Specifies the connection to the X server.

d Specifies the drawable. This is used to determine which screen to create the bitmap on.

data	Specifies the location of the bitmap data.
width	
height	Specify the width and height. These are the dimensions of the bitmap to create.

The XCreateBitmapFromData function allows you to include in your C program (using #include) a bitmap file that was written out by XWriteBitmapFile (X version 11 format only) and, without reading in the bitmap file, The following is an example of getting a gray bitmap:

```
#include "gray.bitmap"
```

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

If insufficient working storage was allocated, XCreateBitmapFromData returns NULL. It is the user's responsibility to free the bitmap using XFreePixmap when done.

10.11. Using the Resource Manager

The resource manager is a database manager but with a twist. In most database systems, you perform a query using an imprecise specification and get back a set of records. The resource manager, however, allows you to specifies a large set of values with an imprecise specification to query the database with a precise specification and get back only a single value. This should be used by applications that need to know what colors, fonts, and other resources the user prefers.

For example, a user of may want to specifies that all windows should have a blue background and all mail reading windows should have a red background. Presuming that all applications use the resource manager, a user can define this information using only two lines of specification. Your personal resource database usually is stored in a file and is loaded onto a server property when you log in. This database is retrieved automatically by Xlib when a connection is opened.

As an example of how the resource manager works, consider a mail reading application called xmh. Assume that it is designed in such a manner that it uses a complex window hierarchy all the way down to individual command buttons, which may be actual small subwindows in some toolkits. These are often called "objects". In such toolkit systems, user interface objects (called "widgets" in the X Toolkit) can be composed of other objects. Each user interface "object" can be assigned a name and a class. Fully qualified names or classes can have arbitrary numbers of component names, but a fully qualified name always has the same number of component names as a fully qualified class. This generally reflects the structure of the application as composed of these objects, starting with the application itself.

For example, the xmh mail program has a name "xmh" and is one of a class of "Mail" programs. By convention, the first character of class components is capitalized, while the first letter of name components is in lowercase. Each name and class finally have an attribute (for example, "foreground" or "font"). If each window is properly assigned a name and class, it becomes easy for the user to specifies attributes of any portion of the application.

At the top level, the application might consist of a paned window (that is, a window divided into several Sections) named "toc". One pane of the paned window is a button box window named "buttons" filled with command buttons. One of these command buttons is used to retrieve ("include") new mail and has the name "include". This window has a fully qualified name "xmh.toc.buttons.include" and a fully qualified class "Xmh.VPaned.Box.Command". Its fully qualified name is the name of its parent, "xmh.toc.buttons", followed by its name "include". Its class is the class of its parent, "Xmh.VPaned.Box", followed by its particular class, "Command". The fully qualified name of a resource is the attribute's name appended to the object's fully qualified name, and the fully qualified class is its class appended to the object's class.

This "include" button needs the following resources:

- Title string
- Font
- Foreground color

- Background color
- Foreground color for its active state
- Background color for its active state

Each of the resources that this button needs are considered to be attributes of the button and, as such, have a name and a class. For example, the foreground color for the button in its active state might be named "activeForeground", and its class would be "Foreground."

When an application looks up a resource (for example, a color), it passes the complete name and complete class of the resource to a lookup routine. After lookup, the resource manager returns the resource value and the representation type.

The resource manager allows applications to store resources by an incomplete specification of name, class, and a representation type, as well as to retrieve them given a fully qualified name and class.

10.11.1. Resource Manager Matching Rules

The algorithm for determining which resource name or names match a given query is the heart of the database. Resources are stored with only partially specified names and classes, using pattern matching constructs. An asterisk ("*") is used to represent any number of intervening components (including none). A dot or period (".") is used to separate adjacent components. All queries fully specifies the name and class of the resource needed. The lookup algorithm then searches the database for the name that most closely matches (is most specific) to this full name and class. The rules (in order of precedence) for a match are:

1. The attribute of the name and class must match. For example, queries for

```
xterm.scrollbar.background          (name)
XTerm.Scrollbar.Background          (class)
```

   will not match the following database entry:

```
xterm.scrollbar:on
```

2. Database entries with name or class prefixed by a dot (".") are more specific than those prefixed by an asterisk ("*"). For example, the entry xterm.geometry is more specific than entry xterm*geometry.
3. Names are more specific than classes. For example, the entry *scrollbar.background is more specific than entry *Scrollbar.Background.
4. A name or class is more specific than omission. For example, the entry Scrollbar*Background is more specific than entry *Background.
5. Left components are more specific than right components. For example, xterm*background is more specific than entry scrollbar*background.
6. If neither a dot (".") nor an asterisk ("*") is specified at the beginning, a dot (".") is implicit. For example, xterm.background is identical to .xterm.background.

As an example of these rules, assume the following user preference specification:

```
xmh*background:                    red
*command.font:                     8x13
*command.background:               blue
*Command.Foreground:               green
xmh.toc*Command.activeForeground:  black
```

A query for the name "xmh.toc.messagefunctions.include.activeForeground" and class "Xmh.VPaned.Box.Command.Foreground" would match "xmh.toc*Command.activeForeground" and return "black". However, it also matches

"*Command.Foreground".

Using the precedence algorithm described above, the resource manager would return the value specified by "xmh.toc*Command.activeForeground".

10.11.2. Basic Resource Manager Definitions

The definitions for the resource manager's use are contained in the <X11/Xresource.h> header file. Xlib also uses the resource manager internally to allow for non-English language error messages.

Database values consist of a size, an address, and a representation type. The size is specified in bytes. The representation type is a way for you to store data tagged by some application-defined type (for example, "font" or "color"). It has nothing to do with the C data type or with its class. The XrmValue structure contains:

```
typedef struct {
        unsigned int size;
        caddr_t addr;
} XrmValue, *XrmValuePtr;
```

A resource database is an opaque type used by the lookup routines.

```
typedef struct _XrmHashBucketRec *XrmDatabase;
```

To initialize the resource manager, use XrmInitialize.

```
void XrmInitialize();
```

Most uses of the resource manager involve defining names, classes, and representation types as string constants. However, always referring to strings in the resource manager can be slow, because it is so heavily used in some toolkits. To solve this problem, a shorthand for a string that is to be used in place of the string when the resource manager is heavily used in many of the resource manager functions. Simple comparisons can be performed rather than string comparisons. The shorthand name for a string is called a "quark" and is the type XrmQuark. On some occasions, you may want to allocate a quark that has no string equivalent.

A quark is to a string what an atom is to a property name in the server, but its use is entirely local to your application.

To allocate a new quark, use XrmUniqueQuark.

```
XrmQuark XrmUniqueQuark()
```

The XrmUniqueQuark function allocates a quark that is guaranteed not to represent any string.

To allocate some memory you will never give back, use

```
char *Xpermalloc(size)
   unsigned int size;
```

The Xpermalloc function is used by some toolkits for permanently allocated storage and allows some performance and space savings over the completely general memory allocator.

Names, classes and representation types are all typedef'd as XrmQuarks.

```
typedef int XrmQuark, *XrmQuarkList;
typedef XrmQuark XrmName;
typedef XrmQuark XrmClass;
typedef XrmQuark XrmRepresentation;
```

Lists are represented as null terminated arrays of quarks. The size of the array must be large enough for the number of components used.

```
typedef XrmQuarkList XrmNameList;
typedef XrmQuarkList XrmClassList;
```

To convert a string to a quark, use XrmStringToQuark.

```
#define XrmStringToName(string) XrmStringToQuark(string)
#define XrmStringToClass(string) XrmStringToQuark(string)
#define XrmStringToRepresentation(string) XrmStringToQuark(string)
```

XrmQuark XrmStringToQuark(*string*)
   char **string*;

string	Specifies the string for which a quark is to be allocated.

To convert a quark to a string, use XrmQuarkToString.

```
#define XrmNameToString(name) XrmQuarkToString(name)
#define XrmClassToString(class) XrmQuarkToString(class)
#define XrmRepresentationToString(type) XrmQuarkToString(type)
```

char *XrmQuarkToString(*quark*)
   XrmQuark *quark*;

quark	Specifies the quark for which the equivalence string is desired.

These routines can be used to convert to and from quark representations. The string pointed to by the return value must not be modified or freed.

To convert a string with one or more components to a quark list, use XrmStringToQuarkList.

```
#define XrmStringToNameList(str, name) XrmStringToQuarkList((str), (name))
#define XrmStringToClassList(str,class) XrmStringToQuarkList((str), (class))
```

void XrmStringToQuarkList(*string*, *quarks_return*)
   char **string*;
   XrmQuarkList *quarks_return*;

string	Specifies the string for which a quark is to be allocated.
quark_return	Returns the list of quarks.

The XrmStringToQuarkList function converts the null terminated string (generally a fully qualified name) to a list of quarks. The components of the string are separated by a "." character.

A binding list is a list of type XrmBindingList and indicates if components of name or class lists are bound tightly or loosely (that is, if wildcarding of intermediate components is specified).

```
typedef enum {XrmBindTightly, XrmBindLoosely} XrmBinding, *XrmBindingList;
```

XrmBindTightly indicates that a dot (".") separates the components, while XrmBindLoosely indicates that an asterisk ("*") separates the components.

To convert a string with one or more components to a binding list and a quark list, use XrmStringToBindingQuarkList.

XrmStringToBindingQuarkList(*string*, *bindings_return*, *quarks_return*)
   char **string*;
   XrmBindingList *bindings_return*;
   XrmQuarkList *quarks_return*;

string	Specifies the string for which a quark is to be allocated.
bindings_return	Returns the binding list. The caller must allocate sufficient space for the binding list before calling XrmStringToBindingQuarkList.
quark_return	Returns the list of quarks. The caller must allocate sufficient space for the quarks list before calling XrmStringToBindingQuarkList.

Component names in the list are separated by a dot (".") or an asterisk ("*") character. If the string does not start with dot or asterisk, a dot (".") is assumed. For example, "*a.b*c" becomes:

quarks	a	b	c
bindings	loose.	tight	loose

Xlib provides resource management functions with which you can manipulate resource databases. The next Sections discuss how to:

- Store and get resources
- Get database levels
- Merge two databases
- Retrieve and store databases

10.11.2.1. Storing Into a Resource Database

To store resources into the database, use XrmPutResource or XrmQPutResource. Both functions take a partial resource specification, a representation type, and a value. This value is copied into the specified database.

The definition for XrmPutResource is:

void XrmPutResource(*database*, *specifier*, *type*, *value*)
  XrmDatabase **database*;
  char **specifier*;
  char **type*;
  XrmValue **value*;

database	Specifies a pointer to the resource database. If database contains NULL, a new resource database is created and a pointer to it is returned in database.
specifier	Specifies a partial specification of the resource.
type	Specifies the type of the resource.
value	Specifies the value of the resource.

XrmPutResource is a convenience function that calls XrmStringToBindingQuarkList followed by:

```
XrmQPutResource(database, bindings, quarks, XrmStringToQuark(type), value)
```

The definition for XrmQPutResource is:

void XrmQPutResource(*database*, *bindings*, *quarks*, *type*, *value*)
  XrmDatabase **database*;
  XrmBindingList *bindings*;
  XrmQuarkList *quarks*;
  XrmRepresentation *type*;
  XrmValue **value*;

database	Specifies a pointer to the resource database. If database contains NULL, a new resource database is created and a pointer to it is returned in database.

bindings	Specifies a list of bindings.
quarks	Specifies the partial name or class list of the resource to be stored.
type	Specifies the type of the resource.
value	Specifies the value of the resource.

To add a resource that is specified as a string, use XrmPutStringResource.

```
void XrmPutStringResource(database, resource, value)
   XrmDatabase *database;
   char *resource;
   char *value;
```

database	Specifies a pointer to the resource database. If database contains NULL, a new resource database is created and a pointer to it is returned in database.
resource	Specifies the resource as a string.
value	Specifies the value of the resource. The value is specified as a string.

XrmPutStringResource adds a resource with the specified value to the specified database. XrmPutStringResource is a convenience routine that takes both the resource and value as strings, converts them to quarks, and then calls XrmQPutResource.

To add a string resource using quarks as a specification, use XrmQPutStringResource.

```
void XrmQPutStringResource(database, bindings, quarks, value)
   XrmDatabase *database;
   XrmBindingList bindings;
   XrmQuarkList quarks;
   char *value;
```

database	Specifies a pointer to the resource database. If database contains NULL, a new resource database is created and a pointer to it is returned in database. If the resource database is NULL, a new database will be created.
bindings	Specifies a list of bindings.
quarks	Specifies the partial name or class list of the resource to be stored.
value	Specifies the value of the resource. The value is specified as a string.

XrmQPutStringResource is a convenience routine that constructs an XrmValue for the value string (by calling strlen, which sets up the address and size) and then calls XrmQPutResource.

To add a single resource entry that is specified as a string that contains both a name and a value, use XrmPutLineResource.

```
void XrmPutLineResource(database, line)
   XrmDatabase *database;
   char *line;
```

database	Specifies a pointer to the resource database. If database contains NULL, a new resource database is created and a pointer to it is returned in database.
line	Specifies the resource value pair as a single string. A single colon (":") separates the name from the value.

XrmPutLineResource adds a single resource entry to the specified database. Any whitespace before or after the name or colon in the line argument is ignored. The value is terminated by a new-line or a NULL character. The value may contain embedded new-line characters prefixed by the "\" and "n" character pair, which are removed before the value is stored in the database. For example, a line might have the value "xterm*background:green\n". Null teminated strings without a new line are also permitted.

10.11.2.2. Looking up from a Resource Database

To retrieve a resource from a resource database, use XrmGetResource or XrmQGetResource.

The definition for XrmGetResource is:

Bool XrmGetResource(*database, str_name, str_class, str_type_return, str_value_return*)
    XrmDatabase *database*;
    char **str_name*;
    char **str_class*;
    char ***str_type_return*;
    XrmValue **value_return*;

database	Specifies the database that is to be used.
str_name	Specifies the fully qualified name of the value being retrieved (as a string).
str_class	Specifies the fully qualified class of the value being retrieved (as a string).
str_type_return	Returns a pointer to the representation type of the destination (as a string).
value_return	Returns the value in the database.

The definition for XrmQGetResource is:

Bool XrmQGetResource(*database, quark_name, quark_class, quark_type_return, value_return*)
    XrmDatabase *database*;
    XrmNameList *quark_name*;
    XrmClassList *quark_class*;
    XrmRepresentation **quark_type_return*;
    XrmValue **value_return*;

database	Specifies the database that is to be used.
quark_name	Specifies the fully qualified name of the value being retrieved (as a quark).
quark_class	Specifies the fully qualified class of the value being retrieved (as a quark).
quark_type_return	Returns a pointer to the representation type of the destination (as a quark).
value_return	Returns the value in the database.

The XrmGetResource and XrmQGetResource functions retrieve a resource from the specified database. Both take a fully qualified name/class pair, a destination resource representation, and the address of a value (size/address pair). The value returned points into database memory; therefore, you must not modify the data.

Currently, the database only frees or overwrites entries on XrmPutResource, XrmQPutResource, or XrmMergeDatabases. A client that is not storing new values into and that is not merging the database should be safe using the address passed back at any time until it exits. If a resource was found, both XrmGetResource and XrmQGetResource return True. Otherwise, they return False.

10.11.2.3. Database Search Lists

Most applications and toolkits do not make random probes into a resource database to fetch resources. The X Toolkit access pattern for a resource database is quite stylized. That is, a series of from one to twenty probes are made with only the last name/class differing in each probe. The XrmGetResource function is at worst a 2^n algorithm, where n is the length of the name/class list. This can be improved upon by the application programmer by prefetching a list of database levels that might match the first part of a name/class list. For example, typedef XrmHashTable *XrmSearchList;

To return a list of database levels, use XrmQGetSearchList.

Bool XrmQGetSearchList(*database*, *names*, *classes*, *list_return*, *list_length*)
  XrmDatabase *database*;
  XrmNameList *names*;
  XrmClassList *classes*;
  XrmSearchList *list_return*;
  int *list_length*;

database	Specifies the database that is to be used.
names	Specifies a list of resource names.
classes	Specifies a list of resource classes.
list_return	Returns a search list for further use. The caller must allocate sufficient space for the list before calling XrmQGetSearchList.
list_length	Specifies the number of entries (not the byte size) allocated for list_return.

The XrmQGetSearchList function takes a list of names and classes and returns a list of database levels where a match might occur. The returned list is in best-to-worst order and uses the same algorithm as XrmGetResource for determining precedence. If list_return was large enough for the search list, XrmQGetSearchList returns True. Otherwise, it returns False.

The size of the search list that must be allocated by the caller is dependent upon the number of levels and wildcards in the resource specifiers that are stored in the database. The worst case length is 3^n, where n is the number of name or class components in names or classes.

When using XrmQGetSearchList followed by multiple probes for resources with a common name and class prefix, only the common prefix should be specified in the name and class list to XrmQGetSearchList.

To search resource database levels for a given resource, use XrmQGetSearchResource.

Bool XrmQGetSearchResource(*list*, *name*, *class*, *type_return*, *value_return*)
  XrmSearchList *list*;
  XrmName *name*;
  XrmClass *class*;
  XrmRepresentation **type_return*;
  XrmValue **value_return*;

list	Specifies the search list returned by XrmQGetSearchList.
name	Specifies the resource name.
class	Specifies the resource class.
type_return	Returns data representation type.
value_return	Returns the value in the database.

The XrmQGetSearchResource function searches the specified database levels for the resource that is fully identified by the specified name and class. The search stops with the first match. XrmQGetSearchResource returns True if the resource was found.

A call to XrmQGetSearchList with a name and class list containing all but the last component of a resource name followed by a call to XrmQGetSearchResource with the last component name and class returns the same database entry as XrmGetResource and XrmQGetResource with the fully qualified name and class.

10.11.2.4. Merging Resource Databases

To merge the contents of one database into another, use XrmMergeDatabases.

```
void XrmMergeDatabases(source_db, target_db)
    XrmDatabase source_db, *target_db;
```

source_db	Specifies the resource database that is to be merged into the target database.
target_db	Specifies a pointer to the resource database into which the source database is to be merged.

The XrmMergeDatabases function merges the contents of one database into another. It may overwrite entries in the destination database. This function is used to combine databases (for example, an application specific database of defaults and a database of user preferences). The merge is destructive; that is, the original database is destroyed.

10.11.2.5. Retrieving and Storing Databases

To retrieve a database from nonvolatile storage, use XrmGetFileDatabase.

```
XrmDatabase XrmGetFileDatabase(filename)
   char *filename;
```

filename	Specifies the resource database file name.

The XrmGetFileDatabase function opens the specified file, creates a new resource database, and loads it with the specifications read in from the specified file. The specified file must contain lines in the format accepted by XrmPutLineResource. If it cannot open the specified file, XrmGetFileDatabase returns NULL.

To store a copy of the application's current database in nonvolatile storage, use XrmPutFileDatabase.

```
void XrmPutFileDatabase(database, stored_db)
   XrmDatabase database;
   char *stored_db;
```

database	Specifies the database that is to be used.
stored_db	Specifies the file name for the stored database.

The XrmPutFileDatabase function stores a copy of the application's current database in the specified file. The file is an ASCII text file that contains lines in the format that is accepted by XrmPutLineResource.

To create a database from a string, use XrmGetStringDatabase.

```
XrmDatabase XrmGetStringDatabase(data)
   char *data;
```

data	Specifies the database contents using a string.

The XrmGetStringDatabase function creates a new database and stores the resources specified in the specified null-terminated string. XrmGetStringDatabase is similar to XrmGetFileDatabase, except that it reads the information out of a string instead of a file. Each line is separated by a new line character in the format accepted by XrmPutLineResource.

10.11.3. Parsing Command Line Options

The XrmParseCommand function can be used to parse the command line arguments to a program and modify a resource database with selected entries from the command line.

```
typedef enum {
        XrmoptionNoArg,              /* Value is specified in OptionDescRec.value */
        XrmoptionIsArg,            /* Value is the option string itself */
        XrmoptionStickyArg,        /* Value is characters immediately following option */
        XrmoptionSepArg,           /* Value is next argument in argv */
        XrmoptionResArg,           /* Resource and value in next argument in argv */
        XrmoptionSkipArg,          /* Ignore this option and the next argument in argv */
        XrmoptionSkipLine          /* Ignore this option and the rest of argv */
} XrmOptionKind;

typedef struct {
        char *option;              /* Option specification string in argv  */
        char *resourceName;        /* Binding and resource name (sans application name)  */
        XrmOptionKind argKind;     /* Which style of option it is  */
        caddr_t value;             /* Value to provide if XrmoptionNoArg  */
} XrmOptionDescRec, *XrmOptionDescList;
```

To load a resource database from a C command line, use XrmParseCommand.

void XrmParseCommand(*db*, *table*, *table_count*, *name*, *argc_return*, *argv_return*,)
  XrmDatabase **db*;
  XrmOptionDescList *table*;
  int *table_count*;
  char **name*;
  int **argc_return*;
  char ***argv_return*;

database	Specifies a pointer to the resource database. If database contains NULL, a new resource database is created and a pointer to it is returned in database.
table	Specifies table of command line arguments to be parsed.
table_count	Specifies the number of entries in the table.
name	Specifies the application name.
argc_return	Before the call, contains the number of arguments. After the call, returns the number of remaining arguments.
argv_return	Before the call, a pointer to the command line arguments. After the call, matched arguments have been removed.

The XrmParseCommand function parses an (arc, argv) pair according to the specified option table, loads recognized options into the specified database, and modifies the (arc, argv) pair to remove all recognized options.

The specified table is used to parse the command line. Recognized entries in the table are removed from argv, and entries are made in the specified resource database. The table entries contain information on the option string, the option name, which style of option and a value to provide if the option kind is XrmoptionNoArg. The argc argument specifies the number of arguments in argv and is set to the remaining number of arguments that were not parsed. The name argument should be the name of your application for use in building the database entry. The name argument is prepended to the resourceName in the option table before storing the specification. No separating (binding) character is inserted. The table must contain either a dot (".") or an asterisk ("*") as the first character in the resourceName entry. To specifies a more completely qualified resource name, the resourceName entry can contain multiple components.

For example, the following is part of the standard option table from the X Toolkit XtInitialize routine:

```
static XrmOptionDescRec opTable[] = {
{"–background",    "*background",          XrmoptionSepArg,    (caddr_t) NULL},
{"–bd",            "*borderColor",         XrmoptionSepArg,    (caddr_t) NULL},
{"-bg",            "*background",                XrmoptionSepArg,    (caddr_t) NULL},
{"-borderwidth",   "*TopLevelShell.borderWidth", XrmoptionSepArg,    (caddr_t) NULL},
{"-bordercolor",   "*borderColor",               XrmoptionSepArg,    (caddr_t) NULL},
{"-bw",            "*TopLevelShell.borderWidth", XrmoptionSepArg,    (caddr_t) NULL},
{"-display",       ".display",                   XrmoptionSepArg,    (caddr_t) NULL},
{"-fg",            "*foreground",                XrmoptionSepArg,    (caddr_t) NULL},
{"-fn",            "*font",                      XrmoptionSepArg,    (caddr_t) NULL},
{"-font",          "*font",                      XrmoptionSepArg,    (caddr_t) NULL},
{"-foreground",    "*foreground",                XrmoptionSepArg,    (caddr_t) NULL},
{"-geometry",      ".TopLevelShell.geometry",    XrmoptionSepArg,    (caddr_t) NULL},
{"-iconic",        ".TopLevelShell.iconic",      XrmoptionNoArg,     (caddr_t) "on"},
{"-name",          ".name",                      XrmoptionSepArg,    (caddr_t) NULL},
{"-reverse",       "*reverseVideo",              XrmoptionNoArg,     (caddr_t) "on"},
{"-rv",            "*reverseVideo",              XrmoptionNoArg,     (caddr_t) "on"},
{"-synchronous",   ".synchronous",               XrmoptionNoArg,     (caddr_t) "on"},
{"-title",         ".TopLevelShell.title",       XrmoptionSepArg,    (caddr_t) NULL},
{"-xrm",           NULL,                         XrmoptionResArg,    (caddr_t) NULL},
};
```

In this table, if the –background (or –bg) option is used to set background colors, the stored resource specifier will match all resources of attribute background. If the –borderwidth option is used, the stored resource specifier applies only to border width attributes of class TopLevelShell (that is, outer-most windows, including pop-up windows). If the –title option is used to set a window name, only the top-most application windows receive the resource.

When parsing the command line, any unique unambiguous abbreviation for an option name in the table is considered a match for the option.

10.12. Using the Context Manager

The context manager provides a way of associating data with a window in your program. Note that this is local to your program; the data is not stored in the server on a property list. Any amount of data in any number of pieces can be associated with a window, and each piece of data has a type associated with it. The context manager requires knowledge of the window ID and type to store or retrieve data.

Essentially, the context manager can be viewed as a two-dimensional, sparse array: one dimension is subscripted by the window ID and the other by a context type field. Each entry in the array contains a pointer to the data. Xlib provides context management functions with which you can save data values, get data values, delete entries, and create a unique context type. The symbols used are in <X11/Xutil.h> header file.

To save a data value that corresponds to a window and context type, use XSaveContext.

int XSaveContext(*display*, *w*, *context*, *data*)
  Display **display*;
  Window *w*;
  XContext *context*;
  caddr_t *data*;

display	Specifies the connection to the X server.
w	Specifies the window with which the data is associated.
context	Specifies the context type to which the data belongs.
data	Specifies the data to be associated with the window and type.

If an entry with the specified window and type already exists, XSaveContext overrides it with the specified context. However, this override has costs in time and space. If you know the entry already exists, it is better to call XDeleteContext first. The XSaveContext function returns a nonzero error code if an error has occurred and zero otherwise. Possible errors are XCNOMEM (out of memory).

To get the data associated with a window and type, use XFindContext.

int XFindContext(*display*, *w*, *context*, *data_return*)
    Display **display*;
    Window *w*;
    XContext *context*;
    caddr_t **data_return*;

display	Specifies the connection to the X server.
w	Specifies the window with which the data is associated.
context	Specifies the context type to which the data belongs.
data_return	Returns a pointer to the data.

Because it is a return value, the data is a pointer. The XFindContext function returns a nonzero error code if an error has occurred and zero otherwise. Possible errors are XCNOENT (context-not-found).

To delete an entry for a given window and type, use XDeleteContext.

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

display	Specifies the connection to the X server.
w	Specifies the window with which the data is associated.
context	Specifies the context type to which the data belongs.

The XDeleteContext function deletes the entry for the given window and type from the data structure. This returns the same error codes that XFindContext returns if called with the same arguments.

To create a unique context type that may be used in subsequent calls to XSaveContext and XFindContext, use XUniqueContext. The definition for this function is:

XContext XUniqueContext()

Appendix A

Xlib Functions and Protocol Requests

The following tables provide a information about Xlib and the X protocol. The first table lists each Xlib function (in alphabetical order) and the corresponding protocol request that it generates. The second table lists each X protocol request (in alphabetical order) and the Xlib functions that reference it.

Xlib Function	Protocol Request
XActivateScreenSaver	ForceScreenSaver
XAddHost	ChangeHosts
XAddHosts	ChangeHosts
XAddToSaveSet	ChangeSaveSet
XAllocColor	AllocColor
XAllocColorCells	AllocColorCells
XAllocColorPlanes	AllocColorPlanes
XAllocNamedColor	AllocNamedColor
XAllowEvents	AllowEvents
XAutoRepeatOff	ChangeKeyboardControl
XAutoRepeatOn	ChangeKeyboardControl
XBell	Bell
XChangeActivePointerGrab	ChangeActivePointerGrab
XChangeGC	ChangeGC
XChangeKeyboardControl	ChangeKeyboardControl
XChangeKeyboardMapping	ChangeKeyboardMapping
XChangePointerControl	ChangePointerControl
XChangeProperty	ChangeProperty
XChangeSaveSet	ChangeSaveSet
XChangeWindowAttributes	ChangeWindowAttributes
XCirculateSubwindows	CirculateWindow
XCirculateSubwindowsDown	CirculateWindow
XCirculateSubwindowsUp	CirculateWindow
XClearArea	ClearArea
XClearWindow	ClearArea
XConfigureWindow	ConfigureWindow
XConvertSelection	ConvertSelection
XCopyArea	CopyArea
XCopyColormapAndFree	CopyColormapAndFree
XCopyGC	CopyGC
XCopyPlane	CopyPlane
XCreateColormap	CreateColormap
XCreateFontCursor	CreateGlyphCursor
XCreateGC	CreateGC
XCreateGlyphCursor	CreateGlyphCursor
XCreatePixmap	CreatePixmap
XCreatePixmapCursor	CreateCursor
XCreateSimpleWindow	CreateWindow
XCreateWindow	CreateWindow
XDefineCursor	ChangeWindowAttributes

Xlib Function	Protocol Request
XDeleteProperty	DeleteProperty
XDestroySubwindows	DestroySubwindows
XDestroyWindow	DestroyWindow
XDisableAccessControl	SetAccessControl
XDrawArc	PolyArc
XDrawArcs	PolyArc
XDrawImageString	ImageText8
XDrawImageString16	ImageText16
XDrawLine	PolySegment
XDrawLines	PolyLine
XDrawPoint	PolyPoint
XDrawPoints	PolyPoint
XDrawRectangle	PolyRectangle
XDrawRectangles	PolyRectangle
XDrawSegments	PolySegment
XDrawString	PolyText8
XDrawString16	PolyText16
XDrawText	PolyText8
XDrawText16	PolyText16
XEnableAccessControl	SetAccessControl
XFetchBytes	GetProperty
XFetchName	GetProperty
XFillArc	PolyFillArc
XFillArcs	PolyFillArc
XFillPolygon	FillPoly
XFillRectangle	PolyFillRectangle
XFillRectangles	PolyFillRectangle
XForceScreenSaver	ForceScreenSaver
XFreeColormap	FreeColormap
XFreeColors	FreeColors
XFreeCursor	FreeCursor
XFreeFont	CloseFont
XFreeGC	FreeGC
XFreePixmap	FreePixmap
XGetAtomName	GetAtomName
XGetFontPath	GetFontPath
XGetGeometry	GetGeometry
XGetIconSizes	GetProperty
XGetImage	GetImage
XGetInputFocus	GetInputFocus
XGetKeyboardContol	GetKeyboardControl
XGetKeyboardMapping	GetKeyboardMapping
XGetMotionEvents	GetMotionEvents
XGetNormalHints	GetProperty
XGetPointerContol	GetPointerControl
XGetPonterMapping	GetPointerMapping
XGetScreenSaver	SetScreenSaver
XGetSelectionOwner	GetSelectionOwner
XGetSizeHints	GetProperty
XGetWMHints	GetProperty
XGetWindowAttributes	GetWindowAttributes
XGetWindowAttributes	GetGeometry

Xlib Function	Protocol Request
XGetWindowProperty	GetProperty
XGetZoomHints	GetProperty
XGrabButton	GrabButton
XGrabKey	GrabKey
XGrabKeyboard	GrabKeyboard
XGrabPointer	GrabPointer
XGrabServer	GrabServer
XInitExtension	QueryExtension
XInstallColormap	InstallColormap
XInternAtom	InternAtom
XKillClient	KillClient
XListExtensions	ListExtensions
XListFonts	ListFonts
XListFontsWithInfo	ListFontsWithInfo
XListHosts	ListHosts
XListInstalledColormaps	ListInstalledColormaps
XListProperties	ListProperties
XLoadFont	OpenFont
XLoadQueryFont	OpenFont
	QueryFont
XLookupColor	LookupColor
XLowerWindow	ConfigureWindow
XMapRaised	ConfigureWindow
	MapWindow
XMapSubwindows	MapSubwindows
XMapWindow	MapWindow
XMoveResizeWindow	ConfigureWindow
XMoveWindow	ConfigureWindow
XNoOp	NoOperation
XOpenDisplay	CreateGC
XParseColor	LookupColor
XPutImage	PutImage
XQueryBestCursor	QueryBestSize
XQueryBestSize	QueryBestSize
XQueryBestStipple	QueryBestSize
XQueryBestTile	QueryBestSize
XQueryColor	QueryColors
XQueryColors	QueryColors
XQueryExtension	QueryExtension
XQueryKeymap	QueryKeymap
XQueryPointer	QueryPointer
XQueryTextExtents	QueryTextExtents
XQueryTextExtents16	QueryTextExtents
XQueryTree	QueryTree
XRaiseWindow	ConfigureWindow
XRecolorCursor	RecolorCursor
XRemoveFromSaveSet	ChangeSaveSet
XRemoveHost	ChangeHosts
XRemoveHosts	ChangeHosts
XReparentWindow	ReparentWindow
XResetScreenSaver	ForceScreenSaver
XResizeWindow	ConfigureWindow

Xlib Function	Protocol Request
XRestackWindows	ConfigureWindow
XRotateBuffers	RotateProperties
XRotateWindowProperties	RotateProperties
XSelectInput	ChangeWindowAttributes
XSendEvent	SendEvent
XSetAccessControl	SetAccessControl
XSetArcMode	ChangeGC
XSetBackground	ChangeGC
XSetClipMask	ChangeGC
XSetClipOrigin	ChangeGC
XSetClipRectangles	SetClipRectangles
XSetCloseDownMode	SetCloseDownMode
XSetCommand	ChangeProperty
XSetDashes	SetDashes
XSetFillRule	ChangeGC
XSetFillStyle	ChangeGC
XSetFont	ChangeGC
XSetFontPath	SetFontPath
XSetForeground	ChangeGC
XSetFunction	ChangeGC
XSetGraphicsExposures	ChangeGC
XSetIconSizes	ChangeProperty
XSetInputFocus	SetInputFocus
XSetLineAttributes	ChangeGC
XSetModifierMapping	SetModifierMapping
XSetNormalHints	ChangeProperty
XSetPlaneMask	ChangeGC
XSetPointerMapping	SetPointerMapping
XSetScreenSaver	SetScreenSaver
XSetSelectionOwner	SetSelectionOwner
XSetSizeHints	ChangeProperty
XSetStandardProperties	ChangeProperty
XSetState	ChangeGC
XSetStipple	ChangeGC
XSetSubwindowMode	ChangeGC
XSetTile	ChangeGC
XSetTSOrigin	ChangeGC
XSetWMHints	ChangeProperty
XSetWindowBackground	ChangeWindowAttributes
XSetWindowBackgroundPixmap	ChangeWindowAttributes
XSetWindowBorder	ChangeWindowAttributes
XSetWindowBorderPixmap	ChangeWindowAttributes
XSetWindowBorderWidth	ConfigureWindow
XSetWindowColormap	ChangeWindowAttributes
XSetZoomHints	ChangeProperty
XStoreBuffer	ChangeProperty
XStoreBytes	ChangeProperty
XStoreColor	StoreColors
XStoreColors	StoreColors
XStoreName	ChangeProperty
XStoreNamedColor	StoreNamedColor
XSync	GetInputFocus

Xlib Function	Protocol Request
XTranslateCoordinates	TranslateCoordinates
XUndefineCursor	ChangeWindowAttributes
XUngrabButton	UngrabButton
XUngrabKey	UngrabKey
XUngrabKeyboard	UngrabKeyboard
XUngrabPointer	UngrabPointer
XUngrabServer	UngrabServer
XUninstallColormap	UninstallColormap
XUnloadFont	CloseFont
XUnmapSubwindows	UnmapSubwindows
XUnmapWindow	UnmapWindow
XWarpPointer	WarpPointer

Protocol Request	Xlib Function
AllocColor	XAllocColor
AllocColorCells	XAllocColorCells
AllocColorPlanes	XAllocColorPlanes
AllocNamedColor	XAllocNamedColor
AllowEvents	XAllowEvents
Bell	XBell
SetAccessControl	XDisableAccessControl
	XEnableAccessControl
	XSetAccessControl
ChangeActivePointerGrab	XChangeActivePointerGrab
SetCloseDownMode	XSetCloseDownMode
ChangeGC	XChangeGC
	XSetArcMode
	XSetBackground
	XSetClipMask
	XSetClipOrigin
	XSetFillRule
	XSetFillStyle
	XSetFont
	XSetForeground
	XSetFunction
	XSetGraphicsExposures
	XSetLineAttributes
	XSetPlaneMask
	XSetState
	XSetStipple
	XSetSubwindowMode
	XSetTile
	XSetTSOrigin
ChangeHosts	XAddHost
	XAddHosts
	XRemoveHost
	XRemoveHosts
ChangeKeyboardControl	XAutoRepeatOff
	XAutoRepeatOn
	XChangeKeyboardControl
ChangeKeyboardMapping	XChangeKeyboardMapping
ChangePointerControl	XChangePointerControl
ChangeProperty	XChangeProperty
	XSetCommand
	XSetIconSizes
	XSetNormalHints
	XSetSizeHints
	XSetStandardProperties
	XSetWMHints
	XSetZoomHints
	XStoreBuffer
	XStoreBytes
	XStoreName
ChangeSaveSet	XAddToSaveSet
	XChangeSaveSet
	XRemoveFromSaveSet

Protocol Request	Xlib Function
ChangeWindowAttributes	XChangeWindowAttributes
	XDefineCursor
	XSelectInput
	XSetWindowBackground
	XSetWindowBackgroundPixmap
	XSetWindowBorder
	XSetWindowBorderPixmap
	XSetWindowColormap
	XUndefineCursor
CirculateWindow	XCirculateSubwindowsDown
	XCirculateSubwindowsUp
	XCirculateSubwindows
ClearArea	XClearArea
	XClearWindow
CloseFont	XFreeFont
	XUnloadFont
ConfigureWindow	XConfigureWindow
	XLowerWindow
	XMapRaised
	XMoveResizeWindow
	XMoveWindow
	XRaiseWindow
	XResizeWindow
	XRestackWindows
	XSetWindowBorderWidth
ConvertSelection	XConvertSelection
CopyArea	XCopyArea
CopyColormapAndFree	XCopyColormapAndFree
CopyGC	XCopyGC
CopyPlane	XCopyPlane
CreateColormap	XCreateColormap
CreateCursor	XCreatePixmapCursor
CreateGC	XCreateGC
	XOpenDisplay
CreateGlphyCursor	XCreateFontCursor
	XCreateGlyphCursor
CreatePixmap	XCreatePixmap
CreateWindow	XCreateSimpleWindow
	XCreateWindow
DeleteProperty	XDeleteProperty
DestroySubwindows	XDestroySubwindows
DestroyWindow	XDestroyWindow
FillPoly	XFillPolygon
ForceScreenSaver	XActivateScreenSaver
	XForceScreenSaver
	XResetScreenSaver
FreeColormap	XFreeColormap
FreeColors	XFreeColors
FreeCursor	XFreeCursor
FreeGC	XFreeGC
FreePixmap	XFreePixmap
GetAtomName	XGetAtomName

Protocol Request	Xlib Function
GetFontPath	XGetFontPath
GetGeometry	XGetGeometry
	XGetWindowAttributes
GetImage	XGetImage
GetInputFocus	XGetInputFocus
	XSync
GetKeyboardControl	XGetKeyboardContol
GetKeyboardMapping	XGetKeyboardMapping
GetMotionEvents	XGetMotionEvents
GetPointerControl	XGetPointerContol
GetPointerMapping	XGetPonterMapping
GetProperty	XFetchBytes
	XFetchName
	XGetIconSizes
	XGetNormalHints
	XGetSizeHints
	XGetWMHints
	XGetWindowProperty
	XGetZoomHints
GetSelectionOwner	XGetSelectionOwner
GetWindowAttributes	XGetWindowAttributes
GrabButton	XGrabButton
GrabKey	XGrabKey
GrabKeyboard	XGrabKeyboard
GrabPointer	XGrabPointer
GrabServer	XGrabServer
ImageText16	XDrawImageString16
ImageText8	XDrawImageString
InstallColormap	XInstallColormap
InternAtom	XInternAtom
KillClient	XKillClient
ListExtensions	XListExtensions
ListFonts	XListFonts
ListFontsWithInfo	XListFontsWithInfo
ListHosts	XListHosts
ListInstalledColormaps	XListInstalledColormaps
ListProperties	XListProperties
LookupColor	XLookupColor
	XParseColor
MapSubwindows	XMapSubwindows
MapWindow	XMapRaised
	XMapWindow
NoOperation	XNoOp
OpenFont	XLoadFont
	XLoadQueryFont
PolyArc	XDrawArc
	XDrawArcs
PolyFillArc	XFillArc
	XFillArcs
PolyFillRectangle	XFillRectangle
	XFillRectangles
PolyLine	XDrawLines

Protocol Request	Xlib Function
PolyPoint	XDrawPoint
	XDrawPoints
PolyRectangle	XDrawRectangle
	XDrawRectangles
PolySegment	XDrawLine
	XDrawSegments
PolyText16	XDrawString16
	XDrawText16
PolyText8	XDrawString
	XDrawText
PutImage	XPutImage
QueryBestSize	XQueryBestCursor
	XQueryBestSize
	XQueryBestStipple
	XQueryBestTile
QueryColors	XQueryColor
	XQueryColors
QueryExtension	XInitExtension
	XQueryExtension
QueryFont	XLoadQueryFont
QueryKeymap	XQueryKeymap
QueryPointer	XQueryPointer
QueryTextExtents	XQueryTextExtents
	XQueryTextExtents16
QueryTree	XQueryTree
RecolorCursor	XRecolorCursor
ReparentWindow	XReparentWindow
RotateProperties	XRotateBuffers
	XRotateWindowProperties
SendEvent	XSendEvent
SetClipRectangles	XSetClipRectangles
SetCloseDownMode	XSetCloseDownMode
SetDashes	XSetDashes
SetFontPath	XSetFontPath
SetInputFocus	XSetInputFocus
SetModifierMapping	XSetModifierMapping
SetPointerMapping	XSetPointerMapping
SetScreenSaver	XGetScreenSaver
	XSetScreenSaver
SetSelectionOwner	XSetSelectionOwner
StoreColors	XStoreColor
	XStoreColors
StoreNamedColor	XStoreNamedColor
TranslateCoordinates	XTranslateCoordinates
UngrabButton	XUngrabButton
UngrabKey	XUngrabKey
UngrabKeyboard	XUngrabKeyboard
UngrabPointer	XUngrabPointer
UngrabServer	XUngrabServer
UninstallColormap	XUninstallColormap
UnmapSubwindows	XUnmapSubWindows
UnmapWindow	XUnmapWindow

Protocol Request	Xlib Function
WarpPointer	XWarpPointer

Appendix B

X Font Cursors

The following are the available cursors and their shapes in fonts.

```
#define XC_num_glyphs 154
#define XC_X_cursor 0
#define XC_arrow 2
#define XC_based_arrow_down 4
#define XC_based_arrow_up 6
#define XC_boat 8
#define XC_bogosity 10
#define XC_bottom_left_corner 12
#define XC_bottom_right_corner 14
#define XC_bottom_side 16
#define XC_bottom_tee 18
#define XC_box_sprial 20
#define XC_center_ptr 22
#define XC_circle 24
#define XC_clock 26
#define XC_coffee_mug 28
#define XC_cross 30
#define XC_cross_reverse 32
#define XC_crosshair 34
#define XC_diamond_cross 36
#define XC_dot 38
#define XC_dot_box_mask 40
#define XC_double_arrow 42
#define XC_draft_large 44
#define XC_draft_small 46
#define XC_draped_box 48
#define XC_exchange 50
#define XC_fleur 52
#define XC_gobbler 54
#define XC_gumby 56
#define XC_hand 58
#define XC_hand1_mask 60
#define XC_heart 62
#define XC_icon 64
#define XC_iron_cross 66
#define XC_left_ptr 68
#define XC_left_side 70
#define XC_left_tee 72
#define XC_leftbutton 74
#define XC_ll_angle 76
#define XC_lr_angle 78
#define XC_man 80
#define XC_middlebutton 82
#define XC_mouse 84
#define XC_pencil 86
#define XC_pirate 88
#define XC_plus 90
#define XC_question_arrow 92
#define XC_right_ptr 94
#define XC_right_side 96
#define XC_right_tee 98
#define XC_rightbutton 100
#define XC_rtl_logo 102
#define XC_sailboat 104
#define XC_sb_down_arrow 106
#define XC_sb_h_double_arrow 108
#define XC_sb_left_arrow 110
#define XC_sb_right_arrow 112
#define XC_sb_up_arrow 114
#define XC_sb_v_double_arrow 116
#define XC_shuttle 118
#define XC_sizing 120
#define XC_spider 122
#define XC_spraycan 124
#define XC_star 126
#define XC_target 128
#define XC_tcross 130
#define XC_top_left_arrow 132
#define XC_top_left_corner 134
#define XC_top_right_corner 136
#define XC_top_side 138
#define XC_top_tee 140
#define XC_trek 142
#define XC_ul_angle 144
#define XC_umbrella 146
#define XC_ur_angle 148
#define XC_watch 150
#define XC_xterm 152
```

Appendix C

Version 10 Compatibility Functions

Drawing and Filling Polygons and Curves

Xlib provides functions with which you can draw or fill arbitrary polygons or curves. These functions are provided mainly for compatibility with X10 and have no server support. That is, they call other Xlib routines, not the server directly. Thus, if you just have straight lines to draw, using XDrawLines or XDrawSegments is much faster.

The functions discussed here provide all the functionality of the X10 routines XDraw, XDrawFilled, XDrawPatterned, XDrawDashed, and XDrawTiled. They are as compatible as possible given X11's new line drawing routines. One thing to note, however, is that VertexDrawLastPoint is no longer supported. Also, the error status returned is the opposite of what it was under X10 (this is the X11 standard error status). XAppendVertex and XClearVertexFlag from X10 also are not supported.

Just how the graphics context you use is set up actually determines whether you get dashes or not, and so on. Lines are properly joined if they connect and include the closing of a closed figure. (See XDrawLines in X 11 for further details.) The functions discussed here fail (return 0) only if they run out of memory or are passed a Vertex list which has a Vertex with VertexStartClosed set which is not followed by a Vertex with VertexEndClosed set.

To achieve the effects of the X10 XDraw, XDrawDashed, and XDrawPatterned, use XDraw.

```
#include <X11/X10.h>

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
vlist	Specifies a pointer to the list of vertices which indicate what to draw.
vcount	Specifies how many vertices are in vlist.

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

Each Vertex, as defined in <X11/Xlib.h>, is a structure with the following elements:

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

The x and y elements are the coordinates of the vertex that are relative to either the upper-left inside corner of the drawable (if VertexRelative is 0) or the previous vertex (if VertexRelative is 1).

The flags, as defined in X10.h, are as follows:

VertexRelative	0x0001	else absolute
VertexDontDraw	0x0002	else draw
VertexCurved	0x0004	else straight
VertexStartClosed	0x0008	else not
VertexEndClosed	0x0010	else not

- If VertexRelative is not set, the coordinates are absolute (relative to the drawable). The first vertex must be an absolute vertex.
- If VertexDontDraw 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 VertexCurved 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 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 a VertexEndClosed bit of 1. The points in between a cycle for the purpose of determining predecessor and successor vertices for the spline algorithm.

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

To achieve the effects of the X10 XDrawTiled and XDrawFilled, use XDrawFilled.

```
#include <X11/X10.h>

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

display	Specifies the connection to the X server.
d	Specifies the drawable.
gc	Specifies the graphics context.
vlist	Specifies a pointer to the list of vertices which indicate what to draw.
vcount	Specifies how many vertices are in vlist.

XDrawFilled draws arbitrary polygons or curves and then fills them.

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

Associating User Data with a Value

These functions have been replaced by the context management functions. See Section 10.12 for further information. It is often necessary to associate arbitrary information with resource IDs. Xlib provides the AssocTable functions with which you can make such an association. Application programs often need to be able to easily refer to their own data structures when an event arrives. The XAssocTable system provides users of the X library with a method of associating their own data structures with X resources. (Bitmaps, Pixmaps, Fonts, Windows, and so on).

An XAssocTable can be used to type X resources. For example, the user may wish to have three or four 'types' of windows each with different properties. This can be accomplished by associating each X window ID with a pointer to a 'window property' data structure defined by the user. A generic type has been defined in the X library for resource IDs. It is called "XId".

There are a few guidelines that should be observed when using an XAssocTable:

- All X IDs are relative to the specified display. Therefore, if you are using multiple displays you need to be sure the correct display is active before performing an XAssocTable operation. XAssocTable imposes no restrictions on the number of X IDs per table, the number of X IDs per display or the number of displays per table.
- Because of the hashing scheme used by the association mechanism the following rules for determining the size of a XAssocTable should be followed. Associations will be made and looked up more efficiently if the table size (number of buckets in the hashing system) is a power of two and if there are not more than 8 XIds per bucket.

To return a pointer to a newly created assoc table, use XCreateAssocTable.

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

size	Specifies the number of buckets in the hash system of XAssocTable.

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.

To create an entry in a specific assoc table, use XMakeAssoc.

```
XMakeAssoc(display, table, x_id, data)
    Display *display;
    XAssocTable *table;
    XID x_id;
    char *data;
```

display	Specifies the connection to the X server.
table	Specifies the assoc table.
x_id	Specifies the X resource ID.
data	Specifies the data to be associated with the X resource ID.

XMakeAssoc inserts data into an XAssocTable keyed on an XId. Data is inserted into the table only once. Redundant inserts are meaningless and cause no problems. The queue in each association bucket is sorted from the lowest XId to the highest XId.

To obtain data from a specific assoc table, use XLookUpAssoc.

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

display	Specifies the connection to the X server.
table	Specifies the assoc table.
x_id	Specifies the X resource ID.

XLookUpAssoc retrieves the data stored in an XAssocTable by its XId. If an appropriately matching XId can be found in the table the routine will return the data associated with it. If the x_id can not be found in the table the routine will return NULL.

To delete an entry from a specific assoc table, use XDeleteAssoc.

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

display	Specifies the connection to the X server.
table	Specifies the assoc table.
x_id	Specifies the X resource ID.

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

To free the memory associated with a specific assoc table, use XDestroyAssocTable.

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

table	Specifies the assoc table.

Using an XAssocTable after it has been destroyed is guaranteed to have unpredictable and probably disastrous consequences!

Appendix D

Extensions

Because X can only evolve by extension to the core protocol, it is important that extensions not be perceivable as second class citizens. At some point, your favorite extensions may be adopted as additional parts of the "X Standard".

Therefore, there should be little to distinguish the use of an extension from that of the core protocol. To avoid having to initialize extensions explicitly in application programs, it is also important that extensions perform "lazy evaluations" and automatically initialize themselves when called for the first time.

This appendix describes techniques for writing extensions to Xlib that will run at essentially the same performance as the core protocol requests.

Note

> It is expected that a given extension to X consists of multiple requests. Defining 10 new features as 10 separate extensions is very bad practice. Rather, they should be packaged into a single extension and should use minor opcodes to distinguish the features.

Basic Protocol Support Routines

The basic protocol requests for extensions are XQueryExtension and XListExtensions.

```
Bool XQueryExtension(display, name, major_opcode, first_event, first_error)
    Display *display;
    char *name;
    int *major_opcode;              /* RETURN */
    int *first_event;               /* RETURN */
    int *first_error;               /* RETURN */
```

XQueryExtension determines if the named extension is present. If so, the major opcode for the extension is returned (if it has one). Otherwise, zero is returned. Any minor opcode and the request formats are specific to the extension. If the extension involves additional event types, the base event type code is returned. Otherwise, zero is returned. The format of the events is specific to the extension. If the extension involves additional error codes, the base error code is returned. Otherwise, zero is returned. The format of additional data in the errors is specific to the extension.

The extension name should be in the ISO LATIN-1 encoding, and upper/lower- case does matter.

```
char **XListExtensions(display, nextensions)
    Display *display;
    int *nextensions;
```

XListExtensions returns a list of all extensions supported by the server.

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

XFreeExtensionList frees the memory allocated by XListExtensions.

Hooking into Xlib

These functions allow one to hook into the library. They are not normally used by application programmers but are used by people who need to extend the core X protocol and the X library interface. The functions, which generate protocol requests for X, are typically called "stubs".

In extensions, stubs first should check to see if they have initialized themselves on a connection. If they have not, they then should call XInitExtension to attempt to initialize themselves on the connection.

If the extension needs to be informed of GC/font allocation or deallocation, or if the extension defines new event types, the functions described in the following sections allow an extension to be called when these events occur.

In <X11/Xlib.h>, the following structure is defined to return the information from XQueryExtension.

```
typedef struct _XExtCodes {              /* public to extension, cannot be changed */
        int extension;                   /* extension number */
        int major_opcode;                /* major op-code assigned by server */
        int first_event;                 /* first event number for the extension */
        int first_error;                 /* first error number for the extension */
} XExtCodes;
```

```
XExtCodes *XInitExtension(display, name)
    Display *display;
    char *name;
```

XInitExtension first calls XQueryExtension to see if the extension exists. Then, it allocates storage for maintaining the information about the extension on the connection, chains this onto the extension list for the connection, and returns the information the stub implementor will need to access the extension.

In particular, the extension number in the XExtCodes structure is needed in other calls below. This extension number is unique only to a single connection.

Hooks into the Library

These functions allow you to define procedures which are to be called when various circumstances occur. The procedures include the creation of a new GC for a connection, the copying of a GC, the freeing a GC, the creating and freeing of fonts, the conversion of events defined by extensions to and from wire format, and the handling of errors.

All of these functions return the previous routine defined for this extension.

```
int (*XESetCloseDisplay(display, extension, proc))()
    Display *display;                    /* display */
    int extension;                       /* extension number */
    int (*proc)();                       /* routine to call when display closed */
```

You use this procedure to define a procedure to be called whenever XCloseDisplay is called. This procedure returns any previously defined procedure, usually NULL.

When XCloseDisplay is called, your routine is called with these arguments:

```
(*proc)(display, codes)
        Display *display;
        XExtCodes *codes;
```

```
int (*XESetCreateGC(display, extension, proc))()
    Display *display;                    /* display */
    int extension;                       /* extension number */
    int (*proc)();                       /* routine to call when GC created */
```

You use this procedure to define a procedure to be called whenever a new GC is created. This procedure returns any previously defined procedure, usually NULL.

When a GC is created, your routine will be called with these arguments:

```
(*proc)(display, gc, codes)
        Display *display;
        GC gc;
        XExtCodes *codes;
```

```
int (*XESetCopyGC(display, extension, proc))()
    Display *display;                    /* display */
    int extension;                       /* extension number */
    int (*proc)();                       /* routine to call when GC copied */
```

You use this procedure to define a procedure to be called whenever a GC is copied. This procedure returns any previously defined procedure, usually NULL.

When a GC is copied, your routine will be called with these arguments:

```
(*proc)(display, gc, codes)
        Display *display;
        GC gc;
        XExtCodes *codes;
```

```
int (*XESetFreeGC(display, extension, proc))()
    Display *display;                    /* display */
    int extension;                       /* extension number */
    int (*proc)();                       /* routine to call when GC freed */
```

You use this procedure to define a procedure to be called whenever a GC is freed. This procedure returns any previously defined procedure, usually NULL.

When a GC is freed, your routine will be called with these arguments:

```
(*proc)(display, gc, codes)
        Display *display;
        GC gc;
        XExtCodes *codes;
```

```
int (*XESetCreateFont(display, extension, proc))()
    Display *display;                    /* display */
    int extension;                       /* extension number */
    int (*proc)();                       /* routine to call when font created */
```

You use this procedure to define a procedure to be called whenever XLoadQueryFont is called. This procedure returns any previously defined procedure, usually NULL.

When XLoadQueryFont is called, your routine will be called with these arguments:

```
(*proc)(display, fs, codes)
        Display *display;
        XFontStruct *fs;
        XExtCodes *codes;
```

```
int (*XESetFreeFont(display, extension, proc))()
    Display *display;                    /* display */
    int extension;                       /* extension number */
    int (*proc)();                       /* routine to call when font freed */
```

You use this procedure to define a procedure to be called whenever XFreeFont is called. This procedure returns any previously defined procedure, usually NULL.

When XFreeFont is called, your routine will be called with these arguments:

```
(*proc)(display, fs, codes)
        Display *display;
        XFontStruct *fs;
        XExtCodes *codes;
```

The next two functions allow you to define new events to the library.

Note

> There is an implementation limit such that your host event structure size cannot be bigger than the size of the XEvent union of structures. There also is no way to guarantee that more than 24 elements or 96 characters in the structure will be fully portable between machines.

```
int (*XESetWireToEvent(display, event_number, proc))()
    Display *display;                   /* display */
    int event_number;                   /* event routine to replace */
    Bool (*proc)();                     /* routine to call when converting event */
```

You use this procedure to define a procedure to be called when an event needs to be converted from wire format (xEvent) to host format (XEvent) form. The event number defines which protocol event number to install a conversion routine for. This procedure returns any previously defined procedure.

Note

> You can replace a core event conversion routine with one of your own, though this is not encouraged. It would, however, allow you to intercept a core event and modify it before being enqueued or otherwise examined.

When Xlib needs to convert an event from wire format to natural host format, your routine will be called with these arguments:

```
Status (*proc)(display, re, event)
        Display *display;
        XEvent *re;
        xEvent event;
```

Your routine must return status to indicate if the conversion suceeded. The re argument is a pointer to where the host format event should be stored, while the event argument is the 32-byte wire event structure. In the XEvent structure you are creating, type must be the first member and window must be the second member. You should fill in the type member with the type specified for the xEvent structure. You should copy all other members from the xEvent structure (wire format) to the XEvent structure (host format). Your conversion routine should return True if the event should be enqueued or False if it should not be enqueued.

```
Status (*XESetEventToWire(display, event_number, proc))()
    Display *display;                   /* display */
    int event_number;                   /* event routine to replace */
    int (*proc)();                      /* routine to call when converting event */
```

You use this procedure to define a procedure to be called when an event needs to be converted from host format (XEvent) to wire format (xEvent) form. The event number defines which protocol event number to install a conversion routine for. This procedure returns any previously defined procedure. It returns zero if the conversion fails or nonzero otherwise.

Note

> You can replace a core event conversion routine with one of your own, though this is not encouraged. It would, however, allow you to intercept a core event and modify it before being sent to another client.

When Xlib needs to convert an event from wire format to natural host format, your routine will be called with these arguments:

```
(*proc)(display, re, event)
        Display *display;
        XEvent *re;
        xEvent event;
```

The re argument is a pointer to the host format event, while the event argument is a pointer to where the 32-byte wire event structure should be stored. In the XEvent structure that you are forming, you must have "type" the first element and "window" the second. You then should fill in the type with the type from the xEvent structure. All other elements then should be copied from the wire format to the XEvent structure.

```
int (*XESetError(display, extension, proc))()
    Display *display;                /* display */
    int extension;                   /* extension number */
    int (*proc)();                   /* routine to call when X error happens */
```

Inside Xlib, there are times that you may want to suppress the calling of the external error handling when an error occurs. This allows status to be returned on a call at the cost of the call being synchronous (though most such routines are query operations, in any case, and are typically programmed to be synchronous).

When Xlib detects an protocol error in _XReply, it will call your procedure with these arguments:

```
int (*proc)(display, err, codes, ret_code)
        Display *display;
        xError *err;
        XExtCodes *codes;
        int *ret_code;
```

The err argument is a pointer to the 32 byte wire format error. The codes argument is a pointer to the extension codes structure. The ret_code argument is the return code you may want _XReply to return.

If your routine returns a value 0, the error is not suppressed, and XError is called. If your routine returns nonzero, the error is suppressed, and _XReply returns the value of ret_code.

```
char *(*XESetErrorString(display, extension, proc))()
    Display *display;                /* display */
    int extension;                   /* extension number */
    char *(*proc)();                 /* routine to call when I/O error happens */
```

```
int (*XESetFlushGC(display, extension, proc))()
    Display *display;                /* display */
    int extension;                   /* extension number */
    char *(*proc)();                 /* routine to call when I/O error happens */
```

The XESetFlushGC procedure is identical to XSetCopyGC, except that XESetFlushGC is called when a GC cache needs to be updated in the server.

The XGetErrorText function returns a string to the user for an error. This procedure allows you to define a routine to be called which should return a pointer to the error message. The following is an example.

```
char *(*proc)(display, code, codes)
        Display *display;
        int code;
        XExtCodes *codes;
```

Your procedure is called with the error code detected. You should return a pointer to a null terminated string containing the error message.

Hooks onto Xlib Data Structures

Various Xlib data structures have provisions for extension routines to chain extension supplied data onto a list. These structures are: GC, Visual, Screen, ScreenFormat, Display, and XFontStruct. Because the list pointer is always the first element in the structure, a single set of routines can be used to manipulate the data on these lists.

The following structure is used in the routines in this section and is defined in <X11/Xlib.h>.

```
typedef struct _XExtData {
        int number;                         /* number returned by XInitExtension */
        struct _XExtData *next;             /* next item on list of data for structure */
        int (*free)();                      /* if defined, called to free private */
        char *private;                      /* data private to this extension. */
} XExtData;
```

When any of the data structures listed above are freed, the list is walked, and the free routine (if any) is called. If free is NULL, then the library will free the data pointed to by private and the structure, itself.

```
XAddToExtensionList(structure, ext_data)
    struct _XExtData **structure;           /* pointer to structure to add */
    XExtData *ext_data;                     /* extension data structure to add */
```

The structure argument is a pointer to one of the data structures enumerated above. You must initialize ext_data->number with the extension number before calling this routine.

It is expected that an extension will add at most one extension data structure to any single data structure's extension data list.

```
XExtData *XFindOnExtensionList(structure, number)
    struct _XExtData **structure;
    int number;                             /* extension number from XInitExtension*/
```

XFindOnExtensionList returns the first extension data structure for the extension numbered number.

The XAllocID macro, which allocates and returns a resource ID, is defined in <X11/Xlib.h>.

```
XAllocIDdisplay)
   Display *display;
```

This macro is a call through the Display structure to the internal resource ID allocator. It returns a resource ID that you can use when creating new resources.

GC Caching

GCs are cached by the library, to allow merging of independent change requests to the same GC into single protocol requests. This is typically called a "write back" cache. Any extension routine whose behavior depends on the contents of a GC must flush the GC cache, to make sure the server has up to date contents in its GC.

The FlushGC macro checks the dirty bits in the library's GC structure and calls _XFlushGCCache if any elements have changed. The FlushGC macro is defined as follows:

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

Note that if you extend the GC to add additional resource ID components, you should ensure that the library stub immediately sends the change request. This is because a client can free a resource immediately after using it, so if you only stored the value in the cache without forcing a protocol request, the resource might be destroyed before being set into the GC. You can use the _XFlushGCCache procedure to force the cache to be flushed. The _XFlushGCCache procedure is defined as follows:

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

Graphics Batching

If you extend X to add more poly graphics primitives, you may be able to take advantage of facilities in the library to allow back-to-back single calls to be transformed into poly requests. This may dramatically improve performance of programs that are not written using poly requests. In the display structure is a pointer to an xReq called last_req which is the last request being processed. By checking that the last request type, drawable, gc, and other options are the same as the new one, and that there is enough space left in the buffer, you may be able to just extend the previous graphics request by extending the length field of the request and appending the data to the buffer. This can cause a 5 times or more improvement in stupid programs. For example, here is the source for the XDrawPoint stub. (Writing extension stubs is discussed in the next section.)

```
#include "copyright.h"

#include "Xlibint.h"

/* precompute the maximum size of batching request allowed */

static int size = sizeof(xPolyPointReq) + EPERBATCH * sizeof(xPoint);

XDrawPoint(dpy, d, gc, x, y)
   register Display *dpy;
   Drawable d;
   GC gc;
   int x, y; /* INT16 */
{
   xPoint *point;
   LockDisplay(dpy);
   FlushGC(dpy, gc);
   {
   register xPolyPointReq *req = (xPolyPointReq *) dpy->last_req;
   /* if same as previous request, with same drawable, batch requests */
   if (
        (req->reqType == X_PolyPoint)
     && (req->drawable == d)
     && (req->gc == gc->gid)
     && (req->coordMode == CoordModeOrigin)
     && ((dpy->bufptr + sizeof (xPoint)) <= dpy->bufmax)
     && (((char *)dpy->bufptr - (char *)req) < size) ) {
      point = (xPoint *) dpy->bufptr;
      req->length += sizeof (xPoint) >> 2;
      dpy->bufptr += sizeof (xPoint);
      }

   else {
     GetReqExtra(PolyPoint, 4, req); /* 1 point = 4 bytes */
     req->drawable = d;
     req->gc = gc->gid;
     req->coordMode = CoordModeOrigin;
     point = (xPoint *) (req + 1);
     }
   point->x = x;
```

```
        point->y = y;
        }
        UnlockDisplay(dpy);
        SyncHandle();
    }
```

There is a limit of EPERBATCH on the number of requests batched, to keep clients from generating very long requests that may monopolize the server. Most of the performance benefit occurs in the first few merged requests. Note that FlushGC is called BEFORE picking up the value of last_req, since it may modify this field.

Writing Extension Stubs

All X requests always contain the length of the request, expressed as a 16-bit quantity of 32-bits. This means that a single request can be no more than 256k Bytes in length. Some servers may not support single requests of such a length. The value of dpy->max_request_size contains the maximum length as defined by the server implementation. For further information, see the X Protocol documentation.

Requests, Replies, and Xproto.h

It may make it easier to understand if you look at the Xproto.h header file. The file contains three sets of definitions that are of interest to the stub implementor: request names, request structures, and reply structures.

You need to generate a file equivalent to Xproto.h for your extension and need to include it in your stub routine. Each stub routine also must include Xlibint.h.

The identifiers are deliberately chosen in such a way that, if the request is called X_DoSomething, then its request structure is xDoSomethingReq, and its reply is xDoSomethingReply. The "GetReq" family of macros, defined in Xlibint.h, takes advantage of this naming scheme.

For each X Request, there is a definition in Xproto.h that looks similar to this:

```
        #define X_DoSomething   42
```

In your extension header file, this will be a minor op-code, instead of a major opcode.

Request Format

Every request contains an 8-bit "major" opcode and a 16-bit length field expressed in units of 4 bytes. Every request consists of 4 bytes of header (containing the major opcode, the length field, and a data byte) followed by zero or more additional bytes of data. The length field defines the total length of the request, including the header. The length field in a request must equal the minimum length required to contain the request. If the specified length is smaller or larger than the required length, the extension should generate a BadLength error. Unused bytes in a request are not required to be zero.

Major opcodes 128 through 255 are reserved for extensions. Extensions are intended to contain multiple requests, so extension requests typically have an additional minor opcode encoded in the "spare" data byte in the request header, but the placement and interpretation of this minor opcode as well as all other fields in extension requests are not defined by the core protocol. Every request is implicitly assigned a sequence number (starting with one) used in replies, errors, and events.

Most protocol requests have a corresponding structure typedef in Xproto.h, which looks like:

```
typedef struct _DoSomethingReq {
        CARD8 reqType;                  /* X_DoSomething */
        CARD8 someDatum;                /* used differently in different requests */
        CARD16 length;                  /* total # of bytes in request, divided by 4 */
        ...
        /* request-specific data */
        ...
} xDoSomethingReq;
```

If a core protocol request has a single 32-bit argument, you need not declare a request structure in your extension header file. Instead, such requests use Xproto.h's xResourceReq structure. This structure is used for any request whose single argument is a Window, Pixmap, Drawable, GContext, Font, Cursor, Colormap, Device, Atom, VisualID, or Time.

```
typedef struct _ResourceReq {
        CARD8 reqType;              /* the request type, e.g. X_DoSomething */
        BYTE pad;                   /* not used */
        CARD16 length;              /* 2 (= total # of bytes in request, divided by 4) */
        CARD32 id;                  /* the Window, Drawable, Font, GContext, etc. */
} xResourceReq;
```

If convenient, you can do something similar in your extension header file.

In both of these structures, the reqType field identifies the type of the request (for example, X_MapWindow or X_CreatePixmap). The length field tells how long the request is in units of 4-byte "longwords". This length includes both the request structure itself and any variable length data, such as strings or lists, that follow the request structure. Request structures come in different sizes, but all requests are padded to be a multiple of 4 bytes long.

A few protocol requests take no arguments at all. Instead, they use Xproto.h's xReq structure, which contains only a reqType and a length (and a pad byte).

If the protocol request requires a reply, then Xproto.h also contains a reply structure typedef:

```
typedef struct _DoSomethingReply {
        BYTE type;                  /* always X_Reply */
        BYTE someDatum;             /* used differently in different requests */
        CARD16 sequenceNumber;      /* # of requests sent so far */
        CARD32 length;              /* # of additional bytes, divided by 4 */
        ...
        /* request-specific data */
        ...
} xDoSomethingReply;
```

Most of these reply structures are 32 bytes long. If there are not that many reply values, then they contain a sufficient number of pad fields to bring them up to 32 bytes. The length field is the total number of bytes in the request minus 32, divided by 4. This length will be nonzero only if:

- The reply structure is followed by variable length data such as a list or string
- The reply structure is longer than 32 bytes

Only GetWindowAttributes, QueryFont, QueryKeymap, and GetKeyboardControl have reply structures longer than 32 bytes.

A few protocol requests return replies that contain no data. Xproto.h does not define reply structures for these. Instead, they use the xGenericReply structure, which contains only a type, length, and sequence number (and sufficient padding to make it 32 bytes long).

Starting to Write a Stub Routine

An Xlib stub routine should always start like this:

```
        #include "Xlibint.h"

        XDoSomething (arguments, ... )
        /* argument declarations */
```

```
{

/* variable declarations, if any */
```

If the protocol request has a reply, then the variable declarations should include the reply structure for the request. The following is an example.

```
xDoSomethingReply rep;
```

Locking Data Structures

Note

Locking has not yet been tested as of February 20, 1988!

In order to lock the display structure, for systems which want to support multithreaded access to a single display connection, each stub will need to lock its critical section. Generally, this section is the point from just before the appropriate GetReq call documented below and when all arguments to the call have been stored into the request. The precise instructions needed for this locking depend upon the machine architecture. Two calls, which are generally implemented as macros, have been provided.

LockDisplay(*display*)
    Display **display*;

UnlockDisplay(*display*)
    Display **display*;

Sending the Protocol Request and Arguments

After the variable declarations, a stub routine should call one of four macros defined in Xlibint.h: GetReq, GetReqExtra, GetResReq, or GetEmptyReq. All of these macros take as their first argument the name of the protocol request as declared in Xproto.h, except with "X_" removed. Each one declares a Display structure pointer, called ''dpy'', and a pointer to a request structure, called ''req'', which is of the appropriate type. The macro then appends the request structure to the output buffer, fills in its type and length field, and sets "req" to point to it.

If the protocol request has no arguments (for instance, X_GrabServer), then use GetEmptyReq:

```
GetEmptyReq (DoSomething);
```

If the protocol request has a single 32-bit argument (such as, a Pixmap, Window, Drawable, Atom, and so on), then use GetResReq. The second argument to the macro is the 32-bit object. X_MapWindow is a good example.

```
GetResReq (DoSomething, rid);
```

The rid argument is the Pixmap, Window, or other resource ID.

If the protocol request takes any other argument list, then call GetReq. After the GetReq, you need to set all the other fields in the request structure, usually from arguments to the stub routine.

```
GetReq (DoSomething);
/* fill in arguments here */
req->arg1 = arg1;
req->arg2 = arg2;
```

A few stub routines (such as, XCreateGC and XCreatePixmap) return a resource ID to the caller but pass a resource ID as an argument to the protocol request. Such routines use the macro XAllocID to allocate a resource ID from the range of IDs that were assigned to this client when it opened the connection.

```
rid = req->rid = XAllocID();
return (rid);
```

Finally, some stub routines transmit a fixed amount of variable-length data after the request. Typically, these are routines (such as, XMoveWindow and XSetBackgroundPixel) are special cases of more general functions like XMoveResizeWindow and XChangeGC. These special case routines use GetReqExtra, which is the same as GetReq, except that it takes an additional argument (the number of extra bytes to allocate in the output buffer after the request structure). This number should always be a multiple of 4.

Variable Length Arguments

Some protocol requests take additional variable length data that follow the xDoSomethingReq structure. The format of these data varies from request to request. Some require a sequence of 8-bit bytes, others a sequence of 16- or 32-bit entities, and still others a sequence of structures.

It is necessary to add the length of any variable length data to the length field of the request structure. That length field is in units of 32-bit longwords. If the data is a string or other sequence of 8-bit bytes, then you must round the length up and shift it before adding:

```
req->length += (nbytes+3)>>2;
```

To transmit the variable length data, use the Data macro. If the data fits into the output buffer, then this macro copies it to the buffer. If it does not fit, however, the Data macro calls _XSend, which transmits first the contents of the buffer and then your data. The Data macro takes three arguments: the Display, a pointer to the beginning of the data, and the number of bytes to be sent.
Data(dpy, (char *) data, nbytes);

If the data are 16-bit entities, then use the PackData macro instead. It takes the same arguments and does the same things, but it does the right thing on machines where a short is 32-bits instead of the usual 16.

Both Data and PackData are macros which may use their last argument more than once, so that argument should be a variable rather than an expression such as ''nitems*sizeof(item)''. You should do that kind of computation in a separate statement before calling Data.

If the protocol request requires a reply, then call the procedure _XSend instead of the Data macro. _XSend takes the same arguments, but, because it sends your data immediately instead of copying it into the output buffer (which would later be flushed anyway by the following call on _XReply), it is faster.

Replies

If the protocol request has a reply, then call _XReply after you have finished dealing with all the fixed and variable length arguments. _XReply flushes the output buffer and waits for an xReply packet to arrive. If any events arrive in the meantime, _XReply enqueues them for later use.

Status _XReply(*display*, *rep*, *extra*, *discard*)
    Display **display*;
    xReply **rep*;
    int *extra*;   /* number of 32-bit words expected after the reply */
    Bool *discard*;   /* should I discard data following "extra" words? */

_XReply waits for a reply packet and copies its contents into the specified rep. _XReply handles error and event packets that occur before the reply is received. _XReply takes four arguments:

- A Display * structure
- A pointer to a reply structure (which must be cast to an xReply *)

- The number of additional bytes (beyond sizeof(xReply) = 32 bytes) in the reply structure
- A boolean which tells _XReply to discard any additional bytes beyond those it was told to read

Because most reply structures are 32 bytes long, the third argument is usually 0. The only exceptions are the replies to GetWindowAttributes, QueryFont, QueryKeymap, and GetKeyboardControl in the core protocol, which have longer replies.

The last argument should be xFalse, if the reply structure is followed by additional variable length data (such as, a list or string). It should be xTrue if there is not any variable length data.

Note

This last argument is provided for upward-compatibility reasons--to allow a client to communicate properly with a hypothetical later version of the server which sends more data than the client expected. For example, some later version of GetWindowAttributes might use a larger, but compatible, xGetWindowAttributesReply which contains additional attribute data at the end.

_XReply returns true, if it received a reply successfully, or false, if it received an XError instead (or some other error condition occurred).

For a request with a reply that is not followed by variable length data, you write something like:

```
_XReply (display, (xReply *)&rep, 0, xTrue);
*ret1 = rep.ret1;
*ret2 = rep.ret2;
*ret3 = rep.ret3;
UnlockDisplay(dpy);
SyncHandle();
return (rep.ret4);
}
```

If there is variable length data after the reply, change the "xTrue" to "xFalse", and use _XRead to read the variable-length data.

Each protocol request is a little different. For further information, see the Xlib sources for examples.

Synchronous Calling

To ease debugging, each routine should have a call to a routine, just before returning to the user, called SyncHandle. This routine generally is implemented as a macro. If synchronous mode is enabled (see XSynchronize), the request is sent immediately. The library, however, waits until any error the routine could generate at the server has been handled.

Allocating and Deallocating Memory

To support the possible reentrancy of these routines, you must observe several conventions when allocating and deallocating memory. This is most often done when returning data to the user from the window system of a size the caller could not know in advance (for example, a list of fonts or a list of extensions). This occurs because the standard C library routines on many systems are not protected against signals or other multithreaded use. The following analogies to standard I/O library routines have been defined:

Xmalloc()	Replaces malloc()
Xfree()	Replaces free()
Xcalloc()	Replaces calloc()

These should be used in place of any calls you would make to the normal C library routines.

If you need a single scratch buffer inside a critical section (for example, to pack and unpack data to and from wire protocol), the general memory allocators may be too expensive for use (particularly in output routines, which are performance critical). The routine below returns a scratch buffer for your use:

```
char *_XAllocScratch(display, nbytes)
    Display *display;
    unsigned long nbytes;
```

This storage must only be used inside of the critical section of your stub.

Portability Considerations

Many machine architectures, including many of the more recent RISC architectures, will not correctly access data at unaligned locations; their compilers will pad out structures to preserve this characteristic. Many other machines capable of unaligned references pad inside of structures as well to preserve alignment, since accessing aligned data is usually much faster. Since the library and the server is using structures to access data at arbitrary points in a byte stream, all data in request and reply packets MUST be naturally aligned; that is 16-bit data start on 16-bit boundaries in the request, 32-bit data on 32-bit boundaries. All requests MUST be a multiple of 32 bits in length, to preserve the natural alignment in the data stream. Pad structures out to 32-bit boundaries. Pad information does not have to be zeroed, unless you wish to preserve such fields for future use in your protocol requests. Floating point varies radically between machines, and should be avoided completely if at all possible.

This code may run on machines with 16-bit ''int''s. So, if any integer argument, variable, or return value either can take only non-negative values or is declared as a CARD16 in the protocol, be sure to declare it as ''unsigned int'' and not as ''int''. (This of course does not apply to booleans or enumerations.)

Similarly, if any integer argument or return value is declared CARD32 in the protocol, declare it as an ''unsigned long'' and not as ''int'' or ''long''. This also goes for any internal variables that may take on values larger than the maximum 16-bit unsigned int.

The library currently assumes that a ''char'' is 8 bits, a ''short'' is 16 bits, an ''int'' is 16 or 32 bits, and a ''long'' is 32 bits. The PackData macro is a half-hearted attempt to deal with the possibility of 32-bit ''shorts''. However, much more work is needed to make this really work properly.

Deriving the Correct Extension Opcode

The remaining problem a writer of an extension stub routine faces that the core protocol does not face is to map from the call to the proper major and minor opcodes. While there are a number of strategies, the simplest and fastest is outlined below.

1. Declare an array of pointers, _NFILE long (this is normally found in <stdio.h> and is the number of file descriptors supported on the system, of type XExtCodes. Make sure these are all initialized to NULL.
2. When your stub is entered, your initialization test is just to use the display pointer passed in to access the file descriptor, and an index into the array. If the entry is NULL, then this is the first time you are entering the routine for this display. Call your initialization routine and pass it the display pointer.
3. Once in your initialization routine, call XInitExtension, and if it succeeds, store the pointer returned into this array. Make sure to establish a close display handler to allow you to zero the entry. Do whatever other initialization your extension requires. (For example, install event handlers, and so on). Your initialization routine would normally return a pointer to the XExtCodes structure for this extension, which is what would normally be found in your array of pointers.
4. After returning from your initialization routine, the stub can now continue normally, since it has its major opcode safely in the XExtCodes structure.

Glossary

Access control list

X maintains a list of hosts from which client programs may be run. By default, only programs on the local host may use the display, plus any hosts specified in an initial list read by the server. This ''access control list'' can be changed by clients on the local host. Some server implementations may also implement other authorization mechanisms in addition to or in place of this mechanism. The action of this mechanism may be conditional based on the authorization protocol name and data received by the server at connection setup.

Active grab

A grab is ''active'' when the pointer or keyboard is actually owned by the single grabbing client.

Ancestors

If W is an inferior of A, then A is an ''ancestor'' of W.

Atom

An ''atom'' is a unique ID corresponding to a string name. Atoms are used to identify properties, types, and selections.

Background

An InputOutput window can have a ''background,'' defined as a pixmap, in which case when regions of the window have their contents lost or invalidated, the server automatically tiles those regions with the background.

Backing store

When a server maintains the contents of a window, the off-screen saved pixels are known as a ''backing store.''

Bit gravity

When a window is resized, the contents of the window are not necessarily discarded. It is possible to request the server (though no guarantees are made) to relocate the previous contents to some region of the window. This attraction of window contents for some location of a window is known as ''bit gravity.''

Bit plane

When a pixmap or window is thought of as a stack of bitmaps, each bitmap is called a ''bit plane'' or ''plane.''

Bitmap

A ''bitmap'' is a pixmap of depth one.

Border

An InputOutput window can have a ''border,'' with equal thickness on all four sides of the window. The contents of the border are defined by a pixmap, and the server automatically maintains the contents of the border. Exposure events are never generated for border regions.

Button grabbing

Buttons on the pointer may be passively ''grabbed'' by a client. When the button is pressed, the pointer is then actively grabbed by the client.

Byte order

For image (pixmap/bitmap) data, byte order is defined by the server, and clients with different native byte ordering must swap bytes as necessary. For all other parts of the protocol, the byte order is defined by the client, and the server swaps bytes as necessary.

Children

The "children" of a window are its first-level subwindows.

Class

Windows can be of different types. See the entries for InputOnly and InputOutput windows for further information about valid window types.

Client

An application program connects to the window system server by some interprocess communication (IPC) path, such as a TCP connection or a shared memory buffer. This program is referred to as a "client" of the window system server. More precisely, the client is the IPC path itself. A program with multiple paths open to the server is viewed as multiple clients by the protocol. Resource lifetimes are controlled by connection lifetimes, not by program lifetimes.

Clipping region

In a graphics context, a bitmap or list of rectangles can be specified to restrict output to a particular region of the window. The image defined by the bitmap or rectangles is called a "clipping region."

Colormap

A "colormap" consists of a set of entries defining color values. The colormap associated with a window is used to display the contents of the window; each pixel value indexes the colormap to produce (red, green, blue) values that drive the guns of a monitor. Depending on hardware limitations, one or more colormaps may be installed at one time, such that windows associated with those maps display with true colors.

Connection

The IPC path between the server and client program is known as a "connection." A client program typically (but not necessarily) has one connection to the server over which requests and events are sent.

Containment

A window "contains" the pointer if the window is viewable and the hotspot of the cursor is within a visible region of the window or a visible region of one of its inferiors. The border of the window is included as part of the window for containment. The pointer is "in" a window if the window contains the pointer but no inferior contains the pointer.

Coordinate system

The coordinate system has X horizontal and Y vertical, with the origin [0, 0] at the upper left. Coordinates are discrete, and in terms of pixels. Each window and pixmap has its own coordinate system. For a window, the origin is at the inside upper left, inside the border.

Cursor

A "cursor" is the visible shape of the pointer on a screen. It consists of a hot spot, a source bitmap, a shape bitmap, and a pair of colors. The cursor defined for a window controls the visible appearance when the pointer is in that window.

Depth

The "depth" of a window or pixmap is the number of bits per pixel it has. The depth of a graphics context is the depth of the drawables it can be used in conjunction with for graphics output.

Device

Keyboards, mice, tablets, track-balls, button boxes, etc., are all collectively known as input ''devices.'' Pointers can have one or more buttons (usually, the most common number is three). The core protocol only deals with two devices: ''the keyboard'' and ''the pointer.''

Direct color

A class of colormap in which a pixel value is decomposed into three separate subfields for indexing. One subfield indexes an array to produce red intensity values; the second subfield indexes a second array to produce blue intensity values; and the third subfield indexes a third array to produce green intensity values. The RGB (red, green, and blue) values in the colormap entry can be changed dynamically.

Display

A server, together with its screens and input devices, is called a ''display.'' The Xlib Display structure contains all information about the particular display and its screens as well as the state that Xlib needs to communicate with the display over a particular connection.

Drawable

Both windows and pixmaps may be used as sources and destinations in graphics operations. These are collectively known as ''drawables.'' However, an InputOnly window cannot be used as a source or destination in a graphics operation.

Event

Clients are informed of information asynchronously by means of ''events.'' These events may be either asynchronously generated from devices, or generated as side effects of client requests. Events are grouped into types. Events are never sent to a client by the server unless the client has specifically asked to be informed of that type of event, but clients ndow described can force events to be sent to other clients. Events are typically reported relative to a window.

Event mask

Events are requested relative to a window. The set of event types a client requests relative to a window is described by using an ''event mask.''

Event synchronization

There are certain race conditions possible when demultiplexing device events to clients (in particular deciding where pointer and keyboard events should be sent when in the middle of window management operations). The event synchronization mechanism allows synchronous processing of device events.

Event propagation

Device-related events ''propagate'' from the source window to ancestor windows until some client has expressed interest in handling that type of event, or until the event is discarded explicitly.

Event source

The smallest window containing the pointer is the ''source'' of a device related event.

Exposure event

Servers do not guarantee to preserve the contents of windows when windows are obscured or reconfigured. ''Exposure'' events are sent to clients to inform them when contents of regions of windows have been lost.

Extension

Named ''extensions'' to the core protocol can be defined to extend the system. Extension to output requests, resources, and event types are all possible, and expected.

Font

A "font" is an array of glyphs (typically characters). The protocol does no translation or interpretation of character sets. The client simply indicates values used to index the glyph array. A font contains additional metric information to determine inter-glyph and inter-line spacing.

Frozen events

Clients can "freeze" event processing while they change the screen.

GC

Shorthand for "graphics context." An equivalent shorthand is GContext. See **graphics context.**

Glyph

A "glyph" is an image, typically of a character, in a font.

Grab

Keyboard keys, the entire keyboard, pointer buttons, the pointer, and the server can be "grabbed" for exclusive use by a client. In general, these facilities are not intended to be used by normal applications, but are intended for various input and window managers to implement various styles of user interfaces.

Graphics context

Various information for graphics output is stored in a "graphics context" ("GC" or "GContext"), such as foreground pixel, background pixel, line width, clipping region, etc. A graphics context can only be used with drawables that have the same root and the same depth as the graphics context.

Gravity

The contents of windows, or subwindows themselves, have a "gravity." This determines how they will be moved when a window ID is resized. See "bit gravity" and "window gravity."

Gray scale

Gray scale can be viewed as a degenerate case of pseudo color, in which case the red, green, and blue values in any given colormap entry are equal, thus producing shades of gray. The gray values can be changed dynamically.

Hotspot

A cursor has an associated "hot spot" which defines a point in the cursor that corresponds to the coordinates reported for the pointer.

Identifier

Each resource has an "identifier," a unique value associated with it that clients use to name the resource. An identifier can be used over any connection to name the resource.

Inferiors

The "inferiors" of a window are all of the subwindows nested below it: the children, the children's children, etc.

Input focus

The "input focus" is where keyboard input goes. Keyboard events are by default sent to the client expressing interest on the window the pointer is in. This is said to be a "real estate driven" input focus. It is also possible to attach the keyboard input to a specific window. Events will then be sent to the appropriate client independent of the pointer position.

Input manager

Control over keyboard input is typically provided by an ''input manager'' client, usually part of a window manager.

InputOnly window

A window that cannot be used for graphics requests. InputOnly windows are ''invisible'' and can be used to control such things as cursors, input event generation, and grabbing. InputOnly windows cannot have InputOutput windows as inferiors.

InputOutput window

The ''normal'' kind of window that is used for both input and output. It usually has a background. InputOutput windows can have both InputOutput and InputOnly windows as inferiors.

Key grabbing

Keys on the keyboard may be passively ''grabbed'' by a client. When the key is pressed, the keyboard is then actively grabbed by the client.

Keyboard grabbing

A client can actively ''grab'' control of the keyboard, and key events will be sent to that client rather than the client the events would normally have been sent to.

Keysym

An encoding of a symbol on a keycap on a keyboard.

Mapped

A window is said to be ''mapped'' if a map call has been performed on it. Unmapped windows and their inferiors are never viewable or visible.

Modifier keys

Shift, Control, Meta, Super, Hyper, Alt, Compose, Apple, CapsLock, ShiftLock, and similar keys are called ''modifier'' keys.

Monochrome

A special case of static gray, in which there are only two colormap entries.

Obscure

A window is ''obscured'' if some other window ''obscures'' it. A window can be partially obscured and still have visible regions.

Obscures

Window A ''obscures'' window B if both are viewable InputOutput windows, if A is higher in the global stacking order, and if the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Note the (fine) distinction with ''occludes.'' Also note that window borders are included in the calculation.

Occlude

A window is ''occluded'' if some other window ''occludes'' it.

Occludes

Window A ''occludes'' window B if both are mapped, if A is higher in the global stacking order, and if the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Note the (fine) distinction with ''obscures.'' Also note that window borders are included in the calculation. Note that InputOnly windows never obscure other windows but can occludes other windows.

Padding

Some padding bytes are inserted in the data stream to maintain alignment of the protocol requests on natural boundaries. This increases ease of portability to some machine architectures.

Parent window

If C is a child of P, then P is the ''parent'' of C.

Passive grab

Grabbing a key or button is a ''passive'' grab. The grab activates when the key or button is actually pressed.

Pixel value

A ''pixel'' is an N-bit value (at a single point), where N is the number of bit planes (that is, the depth of) used in a particular window or pixmap. A pixel in a window indexes a colormap to derive an actual color to be displayed.

Pixmap

A ''pixmap'' is a three dimensional array of bits. A pixmap is normally thought of as a two dimensional array of pixels, where each pixel can be a value from 0 to 2^N-1, where N is the depth (z axis) of the pixmap. A pixmap can also be thought of as a stack of N bitmaps. A pixmap can only be used on the screen that it was created in.

Plane

When a pixmap or window is thought of as a stack of bitmaps, each bitmap is called a ''plane'' or ''bit plane.''

Plane mask

Graphics operations can be restricted to only affect a subset of bit planes of a destination. A ''plane mask'' is a bit mask describing which planes are to be modified, and is stored in a graphics context.

Pointer

The ''pointer'' is the pointing device currently attached to the cursor, and tracked on the screens.

Pointer grabbing

A client can actively ''grab'' control of the pointer, and button and motion events will be sent to that client rather than the client the events would normally have been sent to.

Pointing device

A ''pointing device'' is typically a mouse or tablet, or some other device with effective dimensional motion. Only one visible cursor is defined by the core protocol, and it tracks whatever pointing device is attached as the pointer.

Property

Windows may have associated ''properties,'' consisting of a name, a type, a data format, and some data. The protocol places no interpretation on properties. They are intended as a general-purpose naming mechanism for clients. For example, clients might share information such as resize hints, program names, and icon formats with a window manager by means of properties.

Property list

The ''property list'' of a window is the list of properties that have been defined for the window.

Pseudo color

A class of colormap in which a pixel value indexes the colormap entry to produce independent red, green, and blue values. That is, the colormap is viewed as an array of triples (RGB values). The RGB values can be changed dynamically.

Raise

Changing the stacking order of a window so as to occlude another window is to "raise" that window.

Rectangle

A "rectangle" specified by [x,y,w,h] has an (infinitely thin) outline path with corners at [x,y], [x+w,y], [x+w,y+h] and [x, y+h]. When a rectangle is filled, the lower-right edges are not drawn. For example, if w=h=0, nothing would be drawn. For w=h=1, a single pixel would be drawn.

Redirecting control

Window managers (or client programs) may wish to enforce window layout policy in various ways. When a client attempts to change the size or position of a window, the operation may be "redirected" to a specified client, rather than the operation actually being performed.

Reply

Information requested by a client program by means of the X protocol is sent back to the client with a "reply." Both events and replys are multipexed on the same connection. Most requests do not generate replies. Some requests generate multiple replies.

Request

A command to the server is called a "request." It is a single block of data sent over a connection.

Resource

Windows, pixmaps, cursors, fonts, graphics contexts, and colormaps are known as "resources." They all have unique identifiers associated with them for naming purposes. The lifetime of a resource usually is bounded by the lifetime of the connection over which the resource was created.

RGB values

"Red, green, and blue" intensity values are used to define a color. These values are always represented as 16-bit unsigned numbers, with zero the minimum intensity and 65535 the maximum intensity. The X server scales these values to match the display hardware.

Root

The "root" of a pixmap or graphics context is the same as the root of whatever drawable was used when the pixmap or Gcontext was created. The "root" of a window is the root window under which the window was created.

Root window

Each screen has a "root window" covering it. It cannot be reconfigured or unmapped, but otherwise acts as a full fledged window. A root window has no parent.

Save set

The "save set" of a client is a list of other client's windows which, if they are inferiors of one of the client's windows at connection close, should not be destroyed, and which should be remapped if it is unmapped. Save sets are typically used by window managers to avoid lost windows if the manager should terminate abnormally.

Scanline

A "scanline" is a list of pixel or bit values viewed as a horizontal row (all values having the same y coordinate) of an image, with the values ordered by increasing x coordinate.

Scanline order

An image represented in "scanline order" contains scanlines ordered by increasing y coordinate.

Screen

A server may provide several independent "screens," which typically have physically independent monitors. This would be the expected configuration when there is only a single keyboard and pointer shared among the screens. A Screen structure contains the information about that screen and is linked to the Display structure.

Selection

A "selection" can be thought of as an indirect property with dynamic type. That is, rather than having the property stored in the X server, it is maintained by some client (the "owner"). A selection is global in nature, being thought of as belonging to the user (but maintained by clients), rather than being private to a particular window subhierarchy or a particular set of clients. When a client asks for the contents of a selection, it specifies a selection "target type." This target type can be used to control the transmitted representation of the contents. For example, if the selection is "the last thing the user clicked on," and that is currently an image, then the target type might specify whether the contents of the image should be sent in XYFormat or ZFormat.

The target type can also be used to control the class of contents transmitted, for example, asking for the "looks" (fonts, line spacing, indentation, and so forth) of a paragraph selection, rather than the text of the paragraph. The target type can also be used for other purposes. The semantics are not constrained by the protocol.

Server

The "server," also referred to as the"X server," provides the basic windowing mechanism. It handles IPC connections from clients, demultipexes graphics requests onto the screens, and multiplexes input back to the appropriate clients.

Server grabbing

The server can be "grabbed" by a single client for exclusive use. This prevents processing of any requests from other client connections until the grab is complete. This is typically only a transient state for such things as rubber-banding and pop-up menus, or to execute requests indivisibly.

Sibling

Children of the same parent window are known as "sibling" windows.

Stacking order

Sibling windows may "stack" on top of each other. Windows above both obscure and occlude lower windows. This is similar to paper on a desk. The relationship between sibling windows is known as the "stacking order."

Static color

Static color can be viewed as a degenerate case of pseudo color, in which the RGB values are predefined and read-only. See **pseudo color**.

Static gray

Static gray can be viewed as a degenerate case of gray scale, in which the gray values are predefined and read-only. The values are typically (near-)linear increasing ramps. See **gray scale**.

Stipple

A "stipple pattern" is a bitmap that is used to tile a region to serve as an additional clip mask for a fill operation with the foreground color.

Status

Many Xlib functions return a success status. If the function does not succeed, however, its arguments are not disturbed.

Tile

A pixmap can be replicated in two dimensions to "tile" a region. The pixmap itself is also known as a "tile."

Timestamp

A time value, expressed in milliseconds, typically since the last server reset. Timestamp values wrap around (after about 49.7 days). The server, given its current time is represented by timestamp T, always interprets timestamps from clients by treating half of the timestamp space as being earlier in time than T, and half of the timestamp space as being later in time than T. One timestamp value, represented by the constant CurrentTime is never generated by the server. This value is reserved for use in requests to represent the current server time.

True color

True color can be viewed as a degenerate case of direct color, in which the subfields in the pixel value directly encode the corresponding RGB values. That is, the colormap has predefined read-only RGB values. The values are typically (near-)linear increasing ramps. See **direct color.**

Type

A type is an arbitrary atom used to identify the interpretation of property data. Types are completely uninterpreted by the server. They are solely for the benefit of clients. X predefines type atoms for many frequently used types, and clients also can define new types.

Viewable

A window is "viewable" if it and all of its ancestors are mapped. This does not imply that any portion of the window is actually visible. Graphics requests can be performed on a window when it is not viewable, but output will not be retained unless the server is maintaining backing store.

Visible

A region of a window is "visible" if someone looking at the screen can actually "see" it: the window is viewable and the region is not occluded by any other window.

Window gravity

When windows are resized, subwindows may be repositioned automatically relative to some position in the window. This attraction of a subwindow to some part of its parent is known as "window gravity."

Window manager

Manipulation of windows on the screen, and much of the user interface (policy) is typically provided by a "window manager" client.

XYFormat

The data for a pixmap is said to be in "XYFormat" if it is organized as a set of bitmaps representing individual bit planes.

ZFormat

The data for a pixmap is said to be in "ZFormat" if it is organized as a set of pixel values in scanline order.

Table of Contents

Index

X Toolkit Intrinsics – C Language X Interface

X Window System

X Version 11, Release 2

Joel McCormack

Digital Equipment Corporation
Western Software Laboratory

Paul Asente

Digital Equipment Corporation
Western Software Laboratory

Ralph R. Swick

Digital Equipment Corporation
External Research Group
MIT Project Athena

Table of Contents

Acknowledgments

The design of the X11 Intrinsics was done primarily by Joel McCormack of Digital WSL. Major contributions to the design and implementation also were done by Charles Haynes and Mike Chow of Digital WSL. Additional contributors to the design and/or implementation were:

Paul Asente (Digital WSL)
Rich Hyde (Digital WSL)
Susan Angebranndt (Digital WSL)
Terry Weissman (Digital WSL)
Mary Larson (Digital UEG)
Mark Manasse (Digital SRC)
Jim Gettys (Digital SRC)
Ralph Swick (Project Athena and Digital ERP)
Leo Treggiari (Digital SDT)
Ron Newman (Project Athena)
Mark Ackerman (Project Athena)

Additional contributions to this document were made by Loretta Guarino-Reid (Digital WSL).

The contributors to the X10 toolkit also deserve mention. Although the X11 Intrinsics present an entirely different programming style, they borrow heavily from the implicit and explicit concepts in the X10 toolkit.

The design and implementation of the X10 Intrinsics were done by:

Terry Weissman (Digital WSL)
Smokey Wallace (Digital WSL)
Phil Karlton (Digital WSL)
Charles Haynes (Digital WSL)
Frank Hall (HP)

The design and implementation of the X10 toolkit's sample widgets were by the above, as well as by:

Ram Rao (Digital UEG)
Mary Larson (Digital UEG)
Mike Gancarz (Digital UEG)
Kathleen Langone (Digital UEG)

These widgets provided a checklist of requirements that we had to address in the X11 intrinsics.

Thanks go to Al Mento of Digital's UEG Documentation Group for formatting and generally improving this document and to John Ousterhout of Berkeley for extensively reviewing early drafts of it.

Finally, a special thanks to Mike Chow, whose extensive performance analysis of the X10 toolkit provided the justification to redesign it entirely for X11.

Joel McCormack
Western Software Laboratory
Digital Equipment Corporation

Chapter 1

X Toolkit Overview

The X Toolkit provides the base functionality necessary to build a wide variety of application environments. It is fully extensible and supportive of the independent development of new or extended components. This is accomplished by defining interfaces that mask implementation details from both applications and common component implementors. By following a small set of conventions, a programmer can extend the X Toolkit in new ways and have these extensions function smoothly with the existing facilities.

The X Toolkit is a library package layered on top of the X Window System. This layer extends the basic abstractions provided by X and, thus, provides the next layer of functionality by supplying mechanisms for intercomponent and intracomponent interactions. In the X Toolkit, a widget is a combination of an X window (or subwindow) and its associated semantics.

To the extent possible, the X Toolkit is policy free. The application environment, not the X Toolkit, defines, implements, and enforces:

- Policy
- Consistency
- Style

Each individual widget implementation defines its own policy. The X Toolkit design allows for the development of radically differing widget implementations.

1.1. Introduction

The X Toolkit provides tools that simplify the design of application user interfaces in the X Window System programming environment. It assists application programmers by providing a commonly used set of underlying user-interface functions to manage:

- Toolkit initialization
- Widgets
- Memory
- Window, file, and timer events
- Widget geometry
- Input focus
- Selections
- Resources and resource conversion
- Translation of events
- Graphics contexts
- Pixmaps
- Errors and warnings

At present, the X Toolkit consists of:

- A set of Intrinsic mechanisms for building widgets
- An architectural model for constructing and composing widgets

- A consistent interface (widget set) for programming

The Intrinsics mechanisms are intended for the widget programmer. The architectural model lets the widget programmer design new widgets by using the Intrinsics or by combining other widgets. The application interface layers built on top of the X Toolkit include a coordinated set of widgets and composition policies. Some of these widgets and policies are application domain specific, while others are common across a number of application domains.

The X Toolkit provides an architectural model that is flexible enough to accommodate a number of different application interface layers. In addition, the supplied set of X Toolkit functions are:

- Functionally complete and policy free
- Stylistically and functionally consistent with the X Window System primitives
- Portable across languages, computer architectures, and operating systems

Applications that use the X Toolkit must include the following header files:

- <X11/Xlib.h>
- <X11/Intrinsic.h>
- <X11/StringDefs.h>

and possibly also:

- <X11/Xatoms.h>
- <X11/Shell.h>

Widget implementations should include

- <X11/IntrinsicP.h> instead of <X11/Intrinsic.h>.

The applications should also include the additional headers for each widget class that they are to use (for example, <X11/Label.h> or <X11/Scroll.h>). The Intrinsics object library file is named libXt.a and, on a UNIX-based system, is normally referenced as –lXt.

1.2. Terminology

The following terms are used throughout this manual.

Application programmer

A programmer who uses the X Toolkit to produce an application user interface.

Class

The general group that a specific object belongs to.

Client

A routine that uses a widget in an application or for composing another widget.

Instance

A specific widget object as opposed to a general widget class.

Method

The functions or procedures that a widget itself implements.

Name

The name that is specific to an instance of a widget for a given client.

Object

A software data abstraction consisting of private data and private and public routines that operate on the private data. Users of the abstraction can interact with the object only through calls to the object's public routines. In the X Toolkit, some of the object's public routines are called directly by the application, while others are called indirectly when the application calls the common Routines. In general, if a function is common to all widgets, an application uses a single Intrinsic routine to invoke the function for all types of widgets. If a function is unique to a single widget type, the widget exports the function as another

"Xt" routine.

Resource

A named piece of data in a widget that can be set by a client, by an application, or by user defaults.

User

A person interacting with a workstation.

Widget

An object providing a user-interface abstraction (for example, a Scrollbar widget).

Widget class

The general group that a specific widget belongs to, which is otherwise know as the type of the widget.

Widget programmer

A programmer who adds new widgets to the X Toolkit.

Chapter 2

Widgets

The fundamental data type of the X Toolkit is the widget, which is dynamically allocated and contains state information. Every widget belongs to exactly one ''widget class'' that is statically allocated and initialized and that contains the operations allowable on widgets of that class.

Logically, a widget is a rectangle with associated input/output semantics. Some widgets display information (for example, text or graphics), while others are merely containers for other widgets (for example, a menu box). Some widgets are output-only and do not react to pointer or keyboard input, while others change their display in response to input and can invoke functions that an application has attached to them.

Much of the input/output of a widget is customizable by users. Such customization includes fonts, colors, sizes, border widths, and so on.

A widget instance is composed of two parts:

- A data structure that contains instance-specific values.
- A class structure that contains information that is applicable to all widgets of that class.

Logically, a widget class is the procedures and data that is associated with all widgets belonging to that class. These procedures and data can be inherited by subclasses.

Physically, a widget class is a pointer to a structure. The contents of this structure are constant for all widgets of the widget class, even though the values can vary from widget class to widget class. (Here, ''constant'' means the class structure is initialized at compile-time and never changed, except for a one-shot class initialization and in-place compilation of resource lists, which takes place when the first widget of the class or subclass is created.) A widget instance is allocated and initialized by XtCreateWidget. For further information, see ''Creating Widgets''.

The organization of the declarations and code for a new widget class between a public ''.h'' file, a private ''.h'' file, and the implementation ''.c'' file is described in ''Widget Classes''. The predefined widget classes adhere to these conventions.

2.1. Core Widget Definition

The Core widget contains the definitions of fields common to all widgets. All widgets are subclasses of Core.

2.1.1. CoreClassPart

The common fields for all widget classes are defined in the CoreClassPart structure:

```
typedef struct {
    WidgetClass superclass;                    See "Widget Classes"
    String class_name;                         See "Widget Classes"
    Cardinal widget_size;                      See "Creating Widgets"
    XtProc class_initialize;                   See "Widget Classes"
    XtWidgetClassProc class_part_initialize;   See "Widget Classes"
    Boolean class_inited;                      Private to "XtCreateWidget"
    XtInitProc initialize;                     See "Creating Widgets"
    XtArgsProc initialize_hook;                See "Creating Widgets"
    XtRealizeProc realize;                     See "Creating Widgets"
    XtActionList actions;                      See "Translation Management"
```

```
        Cardinal num_actions;                 See "Translation Management"
        XtResourceList resources;             See "Resource Management"
        Cardinal num_resources;               See "Resource Management"
        XrmClass xrm_class;                   Private to "Resource Management"
        Boolean compress_motion;              See "Mouse Motion Compression"
        Boolean compress_exposure;            See "Exposure Compression"
        Boolean compress_enterleave;          See "Enter/Leave Compression"
        Boolean visible_interest;             See "Widget Exposure and Visibility"
        XtWidgetProc destroy;                 See "Destroying Widgets"
        XtWidgetProc resize;                  See "Geometry Management"
        XtExposeProc expose;                  See "Widget Exposure and Visibility"
        XtSetValuesFunc set_values;           See "Reading and Writing Widget State"
        XtArgsFunc set_values_hook;           See "Reading and Writing Widget State"
        XtAlmostProc set_values_almost;       See "Reading and Writing Widget State"
        XtArgsProc get_values_hook;           See "Reading and Writing Widget State"
        XtWidgetProc accept_focus;            See "Focus Management"
        XtVersionType version;                See "Widget Classes"
        _XtOffsetList callback_private;       Private to "Callbacks"
        String tm_table;                      See "Translation Management"
        XtGeometryHandler query_geometry;     See "Geometry Management"
        } CoreClassPart;
```

All widget classes have the core class fields as their first component. The prototypical type WidgetClass is defined with only this set of fields. Various routines can cast widget class pointers, as needed, to specific widget class types.

```
        typedef struct {
                CoreClassPart core_class;
        } WidgetClassRec, *WidgetClass;
```

The predefined class record and pointer for WidgetClassRec are:

```
                extern WidgetClassRec widgetClassRec;
                extern WidgetClass widgetClass;
```

The opaque types Widget and WidgetClass and the opaque variable widgetClass are defined for generic actions on widgets.

2.1.2. CorePart

The common fields for all widget instances are defined in the CorePart structure:

```
        typedef struct {
        Widget self;
        WidgetClass widget_class;          See "Widget Classes"
        Widget parent;                     See "Widget Classes"
        String name;                       See "Resource Management"
        XrmName xrm_name;                  Private to "Resource Management"
        Screen *screen;                    See "Obtaining Window Information"
        Colormap colormap;                 See "Obtaining Window Information"
        Window window;                        See "Obtaining Window Information"
        Position x;                           See "Geometry Management"
        Position y;                           See "Geometry Management"
        Dimension width;                      See "Geometry Management"
        Dimension height;                     See "Geometry Management"
        Cardinal depth;                       See "Window Attributes"
        Dimension border_width;               See "Geometry Management"
        Pixel border_pixel;                   See "Obtaining Window Information"
        Pixmap border_pixmap;                 See "Obtaining Window Information"
        Pixel background_pixel;               See "Obtaining Window Information"
        Pixmap background_pixmap;             See "Obtaining Window Information"
        _XtEventTable event_table;            Private to "Event Management"
        struct _TMRec tm;                     Private to "Translation Management"
        caddr_t constraints;                  See "Constrained Composite Widgets"
        Boolean visible;                      See "Widget Visibility and Exposure"
        Boolean sensitive;                    See "Setting and Checking Sensitivity"
        Boolean ancestor_sensitive;           See "Setting and Checking Sensitivity"
        Boolean managed;                      See "Composite Widgets"
        Boolean mapped_when_managed;          See "Composite Widgets"
        Boolean being_destroyed;              See "Destroying Widgets"
        XtCallbackList destroy_callbacks;     See "Destroying Widgets"
        WidgetList popup_list;                See"Pop-up Widgets"
        Cardinal num_popups;                  See "Pop-up Widgets"
        } CorePart;
```

All widget instances have the core fields as their first component. The prototypical type Widget is defined with only this set of fields. Various routines can cast widget pointers, as needed, to specific widget types.

```
        typedef struct {
                CorePart core;
        } WidgetRec, *Widget;
```

2.1.3. CorePart Default Values

The default values for the core fields, which are filled in by the Core resource list and the Core initialize procedure, are:

Field	Default Value
self	address of the widget structure (may not be changed)
widget_class	widget_class argument to XtCreateWidget (may not be changed)
parent	parent argument to XtCreateWidget (may not be changed)
name	name argument to XtCreateWidget (may not be changed)
screen	parent's screen, but top-level widget from display specifier (may not be changed)
colormap	the default color map for the screen
window	NULL
x	0
y	0
width	0
height	0
depth	parent's depth, but top-level widget gets root window depth

border_width	1
border_pixel	BlackPixel of screen
border_pixmap	NULL
background_pixel	WhitePixel of screen
background_pixmap	NULL
visible	TRUE
sensitive	TRUE
ancestor_sensitive	bitwise AND of parent's sensitive & ancestor_sensitive
managed	FALSE
map_when_managed	TRUE
being_destroyed	parent's being_destroyed
destroy_callbacks	NULL

2.2. Composite Widget Definition

Composite widgets are a subclass of the Core widget and are more fully described in ''Composite Widgets''.

2.2.1. CompositeClassPart

In addition to the Core widget class fields, Composite widgets have the following class fields:

```
typedef struct {
        XtGeometryHandler geometry_manager;See "Geometry Management"
        XtWidgetProc change_managed;     See "Composite Widgets"
        XtWidgetProc insert_child;       See "Composite Widgets"
        XtWidgetProc delete_child;       See "Composite Widgets"
        XtWidgetProc move_focus_to_next; See "Focus Management"
        XtWidgetProc move_focus_to_prev; See "Focus Management"
} CompositeClassPart;
```

Composite widget classes have the composite fields immediately following the core fields:

```
typedef struct {
        CoreClassPart core_class;
        CompositeClassPart composite_class;
} CompositeClassRec, *CompositeWidgetClass;
```

The predefined class record and pointer for CompositeClassRec are:

```
extern CompositeClassRec compositeClassRec;
extern WidgetClass compositeWidgetClass;
```

The opaque types CompositeWidget and CompositeWidgetClass and the opaque variable compositeWidgetClass are defined for generic operations on widgets that are a subclass of CompositeWidget.

2.2.2. CompositePart

In addition to the CorePart fields, Composite widgets have the following fields defined in the CompositePart structure:

```
typedef struct {
        WidgetList children;                See "Widget Classes"
        Cardinal num_children;              See "Widget Classes"
        Cardinal num_slots;                 See "Composite Widgets"
        Cardinal num_mapped_children;       See "Composite Widgets"
        XtOrderProc insert_position;        See "Creating Widgets"
} CompositePart;
```

Composite widgets have the composite fields immediately following the core fields:

```
typedef struct {
        CorePart core;
        CompositePart composite;
} CompositeRec, *CompositeWidget;
```

2.2.3. CompositePart Default Values

The default values for the composite fields, which are filled in by the Composite resource list and the Composite initialize procedure, are:

Field	Default Value
children	NULL
num_children	0
num_slots	0
num_mapped_children	0
insert_position	internal function InsertAtEnd

2.3. Constraint Widget Definition

Constraint widgets are a subclass of the Composite widget and are more fully described in ''Constrained Composite Widgets''.

2.3.1. ConstraintClassPart

In addition to the Composite class fields, Constraint widgets have the following class fields:

```
typedef struct {
        XtResourceList resources;           See "Constrained Composite Widgets"
        Cardinal num_resources;             See "Constrained Composite Widgets"
        Cardinal constraint_size;           See "Constrained Composite Widgets"
        XtInitProc initialize;              See "Constrained Composite Widgets"
        XtWidgetProc destroy;               See "Constrained Composite Widgets"
        XtSetValuesFunc set_values;         See "Constrained Composite Widgets"
} ConstraintClassPart;
```

Constraint widget classes have the constraint fields immediately following the composite fields:

```
typedef struct {
        CoreClassPart core_class;
        CompositeClassPart composite_class;
        ConstraintClassPart constraint_class;
} ConstraintClassRec, *ConstraintWidgetClass;
```

The predefined class record and pointer for ConstraintClassRec are:

```
        extern ConstraintClassRec constraintClassRec;
        extern WidgetClass constraintWidgetClass;
```

The opaque types ConstraintWidget and ConstraintWidgetClass and the opaque variable constraintWidgetClass are defined for generic operations on widgets that are a subclass of ConstraintWidgetClass.

2.3.2. ConstraintPart

In addition to the CompositePart fields, Constraint widgets have the following fields defined in the ConstraintPart structure:

```
typedef struct { int empty; } ConstraintPart;
```

Constraint widgets have the constraint fields immediately following the composite fields:

```
typedef struct {
        CorePart core;
        CompositePart composite;
        ConstraintPart constraint;
} ConstraintRec, *ConstraintWidget;
```

Chapter 3

Widget Classes

The widget_class field of a widget points to its widget class structure. This structure contains information that is constant across all widgets of that class.

This class-oriented structure means that widget classes do not usually implement directly callable procedures. Rather, they implement procedures that are available through their widget class structure. These class procedures are invoked by generic procedures that envelop common actions around the procedures implemented by the widget class. Such procedures are applicable to all widgets of that class and also to widgets that are subclasses of that class.

All widget classes are a subclass of the Core class and can be subclassed further. Subclassing reduces the amount of code and declarations you write to make a new widget class that is similar to an existing class. For example, you do not have to describe every resource your widget uses in an XtResourceList. Instead, you just describe the resources your widget has that its superclass does not. Subclasses usually inherit many of their superclass's procedures (for example, the expose procedure or geometry handler).

Subclassing can be taken too far. If you create a subclass that inherits none of the procedures of its superclass, you then should consider whether or not you have chosen the most appropriate superclass.

In order to make good use of subclassing, widget declarations and naming conventions are highly stylized. A widget consists of three files:

- A public ".h" file that is used by client widgets or applications
- A private ".h" file used by widgets that are subclasses of the widget
- A ".c" file that implements the widget class

3.1. Widget Naming Conventions

The X Toolkit Intrinsics are merely a vehicle by which programmers can create new widgets and organize a collection of widgets into an application. So that an application need not deal with as many styles of capitalization and spelling as the number of widget classes it uses, the following guidelines should be followed when writing new widgets:

- Use the X naming conventions that are applicable. For example, a record component name is all lower-case and uses underscore (_) for compound words (for example, background_pixmap). Type and procedure names start with upper-case and use capitalization for compound words (for example, XtArgList or XtSetValues).
- A resource name string is spelled identically to the field name, except that compound names use capitalization rather than underscore. To let the compiler catch spelling errors, each resource name should have a macro definition prefixed with XtN. For example, the background_pixmap field has the corresponding resource name identifier XtNbackground-Pixmap, which is defined as the string "backgroundPixmap". Many predefined names are listed in the <X11/StringDefs.h> header file. Before you invent a new name, you should make sure that your proposed name is not already defined or that there already is not name that you can use.
- A resource class string starts with a capital letter, and uses capitalization for compound names (for example,"BorderWidth"). Each resource class string should have a macro definition prefixed with XtC (for example, XtCBorderWidth).
- A resource representation string is spelled identically to the type name (for example, "TranslationTable"). Each representation string should have a macro definition prefixed

with XtR (for example, XtRTranslationTable).

- New widget classes start with a capital and use capitalization for compound words. Given a new class name "AbcXyz" you should derive several names:
  - Partial widget instance structure name AbcXyzPart
  - Complete widget instance structure names AbcXyzRec and _AbcXyzRec
  - Widget instance pointer type name AbcXyzWidget
  - Partial class structure name AbcXyzClassPart
  - Complete class structure names AbcXyzClassRec and _AbcXyzClassRec
  - Class structure variable abcXyzClassRec
  - Class pointer variable abcXyzWidgetClass
- Action procedures available to translation specifications should follow the same naming conventions as procedures. That is, they start with a capital letter and compound names use capitalization. For example, "Highlight" and "NotifyClient".

3.2. Widget Subclassing in Public ".h" Files

The public ".h" file for a widget class is imported by clients and contains:

- A reference to the public ".h" files for the superclass.
- The names and classes of the new resources that this widget adds to its superclass.
- The class record pointer you use to create widget instances.
- The C type you use to declare widget instances of this class.

For example, the following is the public ".h" file for a possible implementation of the Label widget:

```
/* New resources */
#define XtNjustify          "justify"
#define XtNforeground       "foreground"
#define XtNlabel            "label"
#define XtNfont             "font"
#define XtNinternalWidth "internalWidth"
#define XtNinternalHeight"internalHeight"

/* Class record pointer */
extern WidgetClass labelWidgetClass;

/* C Widget type definition */
typedef struct _LabelRec      *LabelWidget;
```

To accommodate operating systems with file name length restrictions, the name of the public ".h" file is the first ten characters of the widget class. For example, the public ".h" file for the Constraint widget is "Constraint.h.".

3.3. Widget Subclassing in Private ".h" Files

The private ".h" file for a widget is imported by widget classes that are subclasses of the widget and contains:

- A reference to the public ".h" file for the class.
- A reference to the private ".h" file for the superclass.

- The new fields that the widget instance adds to its superclass's widget structure.
- The complete widget instance structure for this widget.
- The new fields that this widget class adds to its superclass's Constraint structure, if the widget is a subclass of Constraint.
- The complete Constraint structure, if the widget is a subclass of Constraint.
- The new fields that this widget class adds to its superclass's widget class structure.
- The complete widget class structure for this widget.
- The name of a "constant" of the generic widget class structure.
- For each new procedure in the widget class structure, an "InheritOperation" procedure for subclasses that wish to merely inherit a superclass operation.

For example, the following is the private ".h" file for the Label widget:

```
#include <X11/Label.h>

/* New fields for the Label widget record */
typedef struct {
/* Settable resources */
        Pixel foreground;
        XFontStruct *font;
        String label;                   /* text to display */
        XtJustify justify;
        Dimension internal_width;       /* # of pixels horizontal border */
        Dimension internal_height;      /* # of pixels vertical border */

/* Data derived from resources */
        GC normal_GC;
        GC gray_GC;
        Pixmap gray_pixmap;
        Position label_x;
        Position label_y;
        Dimension label_width;
        Dimension label_height;
        Cardinal label_len;
        Boolean display_sensitive;
} LabelPart;

/* Full instance record declaration */
typedef struct _LabelRec {
        CorePart core;
        LabelPart label;
} LabelRec;

/* New fields for the Label widget class record */
typedef struct { int empty; } LabelClassPart;

/* Full class record declaration */
typedef struct _LabelClassRec {
        CoreClassPart core_class;
        LabelClassPart label_class;
} LabelClassRec;
```

```
/* Class record variable */
extern LabelClassRec labelClassRec;
```

To accommodate operating systems with file name length restrictions, the name of the private ".h" file is the first nine characters of the widget class followed by a capital "P". For example, the private ".h" file for the Constraint widget is "ConstrainP.h.".

3.4. Widget Subclassing in ".c" Files

The ".c" file for a widget contains the structure initializer for the class record variable. This initializer can be broken up into several parts:

- Class information (for example, superclass, class_name, widget_size, class_initialize, class_inited).
- Data Constants (for example, resources and num_resources, actions and num_actions, visible_interest, compress_motion, compress_exposure, version).
- Widget Operations (for example, initialize, realize, destroy, resize, expose, set_values, accept_focus, and any operations specific to the widget).

The superclass field points to the superclass WidgetClass record. For direct subclasses of the generic core widget, superclass should be initialized to the address of the widgetClassRec structure. The superclass is used for class chaining operations and for inheriting or enveloping a superclass's operations. (See "Superclass Chaining," "Inheriting Superclass Operations," and "Calling Superclass Operations.")

The class_name field contains the text name for this class (used by the resource manager). For example, the Label widget has the string "Label".

The widget_size field is the size of the corresponding Widget structure (not the size of the Class structure).

The version field indicates the toolkit version number and is used for runtime consistency checking of the X Toolkit and widgets in an application. Widget writers must set it to the symbolic value XtVersion in the widget class initialization.

All other fields are described in their respective sections.

The following is a somewhat compressed version of the ".c" file for the Label widget. (The "resources" table is described in the section "Resource Management").

```
/* Resources specific to Label */
#define XtRjustify      "Justify"
static XtResource resources[] = {
        {XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),
          XtOffset(LabelWidget, label.foreground), XtRString, "Black"},
        {XtNfont,  XtCFont, XtRFontStruct, sizeof(XFontStruct *),
          XtOffset(LabelWidget, label.font),XtRString, "Fixed"},
        {XtNlabel,  XtCLabel, XtRString, sizeof(String),
          XtOffset(LabelWidget, label.label), XtRString, NULL},
                        .
                        .
                        .
}

/* Forward declarations of procedures */
static void ClassInitialize();
static void Initialize();
static void Realize();
```

```
        .
        .
        .

/* Class record constant */
LabelClassRec labelClassRec = {
 {
  /* core_class fields */
     /* superclass           */          (WidgetClass) &widgetClassRec,
     /* class_name           */          "Label",
     /* widget_size          */          sizeof(LabelRec),
     /* class_initialize     */          ClassInitialize,
     /* class_part_initialize */         NULL,
     /* class_inited         */          FALSE,
     /* initialize           */          Initialize,
     /* initialize_hook      */          NULL,
     /* realize              */          Realize,
     /* actions              */          NULL,
     /* num_actions          */          0,
     /* resources            */          resources,
     /* num_resources        */          XtNumber(resources),
     /* xrm_class            */          NULLQUARK,
     /* compress_motion      */          TRUE,
     /* compress_exposure                */TRUE,
     /* compress_enterleave              */TRUE,
     /* visible_interest     */          FALSE,
     /* destroy              */          NULL,
     /* resize               */          Resize,
     /* expose               */          Redisplay,
     /* set_values           */          SetValues,
     /* set_values_hook      */          NULL,
     /* set_values_almost    */          XtInheritSetValuesAlmost,
     /* get_values_hook      */          NULL,
     /* accept_focus         */          NULL,
     /* version              */          XtVersion,
     /* callback_offsets     */          NULL,
     /* tm_table             */          NULL
 }
};

/* Class record pointer */
WidgetClass labelWidgetClass = (WidgetClass) &labelClassRec;

/* Private procedures */
        .
        .
        .
```

3.5. Initialization of a Class: the class_initialize and class_part_initialize procedures

Many class records can be initialized completely at compile time. But in some cases, a class may want to register type converters or perform other sorts of ''one-shot'' initialization.

Because the C language does not have initialization procedures that are invoked automatically when a program starts up, a widget class can declare a class_initialize procedure that will be automatically called exactly once by the X Toolkit. A class initialization procedure is of type XtProc:

```
typedef void (*XtProc)();

void Proc()
```

A widget class indicates that it has no class initialization procedure by specifying NULL in the class_initialize field.

In addition to doing class initializations that get done exactly once, some classes need to perform additional initialization for fields in its part of the class record. These get done not just for the particular class but for subclasses as well. This is done in the class's class part initialization procedure. The class part initialization procedure is of type XtClassProc:

```
typedef void (*XtClassProc)();

void ClassProc(widgetClass)
    WidgetClass widgetClass;
```

During class initialization, the class part initialization procedure for the class and all its superclasses are called in a superclass to subclass order on the class record. These procedures have the responsibility of doing any dynamic initializations necessary to their class's part of the record. The most common is the resolution of any inherited methods defined in the class. For example, if a widget class C has superclasses Core, Composite, A, and B, the class record for C first is passed to Core's class_part_initialize record. This resolves any inherited core methods and compiles the textual representations of the resource list and action table that are defined in the class record. Next, the Composite's class_part_initialize is called to initialize the composite part of C's class record. Finally, the class_part_initialize procedures for A, B, and C (in order) are called. For further information, see ''Inheriting Superclass Operations''. Classes that do not define any new class fields or that need no extra processing for them can specify NULL in the class_part_initialize field.

All widget classes (whether they have a class initialization procedure or not) should start off with their class_inited field FALSE.

The first time a widget of that class is created, XtCreateWidget ensures that the widget class and all superclasses are initialized, in superclass to subclass order, by checking each class_inited field and, if it is FALSE, calling the class_initialize and the class_part_initialize procedures for the class and all its superclasses. The class_inited field is then set to TRUE. After the one-time initialization, a class structure is constant.

The following provides the class initialization procedure for Label.

```
static void ClassInitialize()
{
        XtQEleft   = XrmStringToQuark("left");
        XtQEcenter = XrmStringToQuark("center");
        XtQEright  = XrmStringToQuark("right");

        XtAddConverter(XtRString, XtRJustify, CvtStringToJustify, NULL, 0);
}
```

3.6. Obtaining the Class and Superclass of a Widget

To obtain the class of a widget, use XtClass.

```
WidgetClass XtClass(w)
    Widget w;
```

w	Specifies the widget.

XtClass returns a pointer to the widget's class structure.

To obtain the superclass of a widget, use XtSuperclass.

```
WidgetClass XtSuperclass(w)
    Widget w;
```

w	Specifies the widget.

XtSuperclass returns a pointer to the widget's superclass class structure.

3.7. Verifying the Subclass of a Widget

To check the subclass that a widget belongs to, use XtIsSubclass.

```
Boolean XtIsSubclass(w, widget_class)
    Widget w;
    WidgetClass widget_class;
```

w	Specifies the widget under question.
widget_class	Specifies the widget class to test against.

XtIsSubclass returns TRUE if the class of the specified widget w is equal to or is a subclass of widget_class. The specified widget w may be arbitrarily far down the subclass chain; it need not be an immediate subclass of widget_class. Composite widgets that wish to restrict the class of the items they contain can use XtIsSubclass to find out if a widget belongs to the desired class of objects.

To check the subclass that a widget belongs to and to generate a debugging error message, use XtCheckSubclass.

```
void XtCheckSubclass(w, widget_class, message)
    Widget w;
    WidgetClass widget_class;
    String message;
```

w	Specifies the widget under question.
widget_class	Specifies the widget class to test against.
message	Specifies an error message.

XtCheckSubclass determines if the class of the specified w is equal to or is a subclass of widget_class. Again, w may be any number of subclasses down the chain and need not be an immediate subclass of widget_class. If w is not a subclass, XtCheckSubclass constructs an error message from the supplied message, the widget's actual class, and the expected class. Then, it calls XtError. XtCheckSubclass should be used at the entry-point of exported routines to ensure that the client has passed in a valid widget class for the exported operation.

XtCheckSubclass is only executed when including and linking against the debugging version of the Intrinsics. Otherwise, it is defined as the empty string and so generates no code.

3.8. Superclass Chaining

Some fields defined in the widget class structure are self-contained and are independent of the values for these fields defined in superclasses. Among these are:

- class_name
- widget_size
- realize
- visible_interest
- resize
- expose
- accept_focus
- compress_motion
- compress_exposure
- compress_enterleave
- set_values_almost
- tm_table
- version

Some fields defined in the widget class structure make sense only after their superclass has been operated on. In this case, the invocation of a single operation actually first accesses the Core class, then the subclass, and so on down the class chain to the widget class of the widget. These superclass-to-subclass fields are:

- class_initialize
- class_part_initialize
- initialize_hook
- set_values_hook
- get_values_hook
- initialize
- set_values
- resources

For subclasses of Constraint, the constraint resources field is chained from the Constraint class down to the subclass.

Some fields defined in the widget class structure make sense only after their subclass has been operated on. In this case, the invocation of a single operation actually first accesses the widget class, then its superclass, and so on up the class chain to the Core class. The subclass-to-superclass fields are:

- destroy
- actions

3.9. Inheriting Superclass Operations

A widget class is free to use any of its superclass's self-contained operations rather than implementing its own code. The most frequently inherited operations are:

- expose

- realize
- insert_child
- delete_child
- geometry_manager

To inherit an operation "xyz", you simply specify the procedure XtInheritXyz in your class record.

Every class that declares a new procedure in its widget class part must provide for inheriting the procedure in its class_part_initialize procedure. (The special chained operations initialize, set_values, and destroy declared in the Core record do not have inherit procedures. Widget classes that do nothing beyond what their superclass does for these procedures just specify NULL for the procedure in their class records.)

Inheriting works by comparing the value of the field with a known, special value and by copying in the superclass's value for that field if a match occurs. This special value is usually the intrinsic routine _XtInherit cast to the appropriate type.

For example, the Composite class's private include file contains these definitions:

```
#define XtInheritGeometryManager ((XtGeometryHandler) _XtInherit)
#define XtInheritChangeManaged ((XtWidgetProc) _XtInherit)
#define XtInheritInsertChild ((XtArgsProc) _XtInherit)
#define XtInheritDeleteChild ((XtWidgetProc) _XtInherit)
#define XtInheritMoveFocusToNext ((XtWidgetProc) _XtInherit)
#define XtInheritMoveFocusToPrev ((XtWidgetProc) _XtInherit)
```

The Composite's class_part_initialize procedure begins:

```
static void CompositeClassPartInitialize(widgetClass)
        WidgetClass widgetClass;
{
    register CompositeWidgetClass wc = (CompositeWidgetClass) widgetClass;
    CompositeWidgetClass super = (CompositeWidgetClass) wc->core.class.superclass

    if (wc->composite_class.geometry_manager == XtInheritGeometryManager) {
       wc->composite_class.geometry_manager = super->composite_class.geometry_man
    }

    if (wc->composite_class.change_managed == XtInheritChangeManaged) {
       wc->composite_class.change_managed = super->composite_class.change_manage
    }
           .
           .
           .
```

The inherit procedures defined for Core are:

- XtInheritRealize
- XtInheritResize
- XtInheritExpose
- XtInheritSetValuesAlmost
- XtInheritAcceptFocus

The inherit procedures defined for Composite are:

- XtInheritGeometryManager
- XtInheritChangeManaged
- XtInheritInsertChild
- XtInheritDeleteChild
- XtInheritMoveFocusToNext
- XtInheritMoveFocusToPrev

3.10. Calling Superclass Operations

A widget class sometimes explicitly wants to call a superclass operation that normally is not chained. For example, a widget's expose procedure might call its superclass's expose and then perform a little more work of its own. Composite classes with fixed children can implement insert_child by first calling their superclass's insert_child procedure and then calling XtManageChild to add the child to the managed list.

Note that the class procedure should call its own superclass procedure, not the widget's superclass procedure. That is, it should use its own class pointers only, not the widget's class pointers. This technique is referred to as ''enveloping'' the superclass's operation.

The following is abbreviated code for a possible implementation of a Shell's insert_child procedure:

```
static void InsertChild(w)
        Widget w;
{
        (*(((CompositeWidgetClass)XtSuperclass(shellWidgetClass))
                ->composite_class.insert_child)) (w);
        XtManageChild(w);                    /* Add to managed set now */
}
```

Chapter 4

Instantiating Widgets

Widgets are either "primitive" or "composite". Either kind of widget can have "pop-up" children widgets, but only composite widgets can have "normal" children widgets. A composite widget may in unusual circumstances have zero normal children but usually has at least one. Widgets with no children of any kind are leaves of a widget tree. Widgets with one or more children are intermediate nodes of a tree. The shell widget returned by XtInitialize or XtCreateApplicationShell is the root of a widget tree.

The "normal" children of the widget tree exactly duplicates the associated window tree. Each pop-up child has a window which is a child of the root window so that the pop-up window is not clipped. Again, the normal children of a pop-up exactly duplicates the window tree associated with the pop-up window.

A widget tree is manipulated by several X Toolkit functions. For example, XtRealizeWidget traverses the tree downward to recursively realize normal children widgets. XtDestroyWidget traverses the tree downward to destroy all children. The functions that fetch and modify resources traverse the tree upward to determine the inheritance of resources from a widget's ancestors. XtMakeGeometryRequest traverses the tree one level upward to get the geometry manager responsible for a normal widget child's geometry.

To facilitate up-traversal of the widget tree, each widget has a pointer to its parent widget. Shell widgets returned by XtInitialize and XtCreateApplicationShell have a parent pointer of NULL.

To facilitate down-traversal of the widget tree, each composite widget has a pointer to an array of children widgets. This array includes all normal children created, not just the subset of children that are managed by the composite widget's geometry manager.

In addition, every widget has a pointer to an array of pop-up children widgets.

4.1. Initializing the X Toolkit

Before any of the X Toolkit functions can be called by the application, it must initialize the toolkit.

To initialize the X Toolkit, the application must call the XtInitialize function.

```
Widget XtInitialize(shell_name, application_class, options, num_options, argc, argv)
    String shell_name;
    String application_class;
    XrmOptionDescRec options[];
    Cardinal num_options;
    Cardinal *argc;
    String argv[];
```

shell_name Specifies the name of the application shell widget instance, which usually is something generic like "main".

application_class
Specifies the class name of this application, which usually is the generic name for all instances of this application. By convention, the class name is formed by reversing the case of the application's first two letters. For example, an application named "xterm" would have a class name of "XTerm".

options Specifies how to parse the command line for any application-specific resources. The options argument is passed as a parameter to XrmParseCommand. For

	further information, see *Xlib – C Language X Interface.*
num_options	Specifies the number of entries in options list.
argc	Specifies a pointer to the number of command line parameters.
argv	Specifies the command line parameters.

XtInitialize builds the resource database, parses the command line, opens the display, and initializes the X Toolkit. It returns a TopLevelShell widget to use as the parent of the application's root widget.

XtInitialize modifies argc and argv to contain just the parameters that were not a display, geometry, or resource specification. If the modified argc is not zero (0), most applications simply print out the modified argv along with a message about the allowable options.

An application can have multiple top-level widgets. The widget returned by XtInitialize has the WM_COMMAND property set for session managers. See "Shell Widgets" for more information.

XtInitialize saves the application name and class_name for qualifying all widget resource specifiers. On UNIX-based systems, the application name is the final component of argv[0]. (This can be modified from the command line by specifying the –name option.) The application name and class_name are used as the left-most components in all widget resource names for this application.

4.2. Loading the Resource Database

XtInitialize loads the application's resource database from three sources in the following order:

- Application-specific class resource file
- Server resource file
- User's environment resource file

The application-specific resource file name is constructed from the class name of the application and points to a site-specific resource file that usually is installed by the site manager when the application is installed. On UNIX-based systems, the application resource file is /usr/lib/X11/app-defaults/*class*, where class is the application class name.

The server resource file is the contents of the X server's RESOURCE_MANAGER property, as returned by XOpenDisplay. If no such property exists for the display, the .Xdefaults file in the user's home directory, if it exists, is loaded in place of the server property.

The user's environment resource file name is constructed by using the value of the user's XENVIRONMENT variable for the full path of the file. If this environment variable does not exist, XtInitialize looks in the user's home directory for the .Xdefaults-host file, where host is the name of the user's host machine. If the resulting resource file exists, XtInitialize loads its contents into the resource database.

4.3. Parsing the Command Line

XtInitialize first parses the command line looking for the following options:

–display	Specifies the display name for XOpenDisplay.
–synchronize	Calls XSynchronize to put Xlib into synchronous mode.
–name	Sets the resource name prefix in place of argv[0].

XtInitialize has a table of standard command line options for adding resources to the resource database, and it takes as a parameter additional application-specific resource abbreviations. The format of this table is:

```
typedef enum {
        XrmoptionNoArg,          /* Value is specified in OptionDescRec.value   */
        XrmoptionIsArg,          /* Value is the option string itself   */
        XrmoptionStickyArg,      /* Value is characters immediately following optior
        XrmoptionSepArg,         /* Value is next argument in argv   */
        XrmoptionSkipArg,        /* Ignore this option and the next argument in arg\
        XrmoptionSkipLine        /* Ignore this option and the rest of argv   */
} XrmOptionKind;

typedef struct {
        char *option;            /* Option name in argv   */
        char *specifier;         /* Resource name (sans application name)   */
        XrmOptionKind argKind;   /* Which style of option it is   */
        caddr_t value;           /* Value to provide if XrmoptionNoArg   */
} XrmOptionDescRec, *XrmOptionDescList;
```

The standard table contains the following entries:

Option string	Resource name	Argument Kind	Resource value
–background	background	SepArg	next argument
–bd	borderColor	SepArg	next argument
–bg	background	SepArg	next argument
–borderwidth	borderWidth	SepArg	next argument
–bordercolor	borderColor	SepArg	next argument
–bw	borderWidth	SepArg	next argument
–display	display	SepArg	next argument
–fg	foreground	SepArg	next argument
–fn	font	SepArg	next argument
–font	font	SepArg	next argument
–foreground	foreground	SepArg	next argument
–geometry	geometry	SepArg	next argument
–name	name	SepArg	next argument
–reverse	reverseVideo	NoArg	on [not implemented]
–rv	reverseVideo	NoArg	on [not implemented]
+rv	reverseVideo	NoArg	off [not implemented]
–synchronize	synchronize	NoArg	on
–title	title	SepArg	next argument
–xrm	next argument	ResArg	next argument

Notes

1. Any unique abbreviation for an option name in the standard table or in the application table is accepted.

2. For backwards compatibility with older command line syntax, an X Toolkit installation (compile time) option allows the following arguments on the command line:

=	geometry	IsArg	this argument
:	display	IsArg	this argument

The colon (:) can appear anywhere within the argument, and the argument will be accepted as the display string, if the –display argument is not specified on the command line.

The –xrm option provides a method of setting any resource in an application. The next argument should be a quoted string identical in format to a line in the user resources file. For example, to give a red background to all command buttons in an application named xmh, you can start it up as:

```
xmh –xrm 'xmh*Command.background: red'
```

When it fully parses the command line, XtInitialize merges the application option table with the standard option table and then calls the Xlib XrmParseCommand function. An entry in the application table with the same name as an entry in the standard table over-rides the standard table entry. If an option name is a prefix of another option name, both names are kept in the merged table. Although option tables need not be sorted by option name, XrmParseCommand is somewhat more efficient if they are.

4.4. Obtaining Window Information from a Widget

The Core widget definition contains the screen and window IDs. The window field may be NULL for a while (see "Creating Widgets" and "Realizing Widgets").

The display pointer, the parent widget, screen pointer, and window of a widget are returned by the following macros:

```
Display *XtDisplay(w)
    Widget w;
```

```
Widget XtParent(w)
    Widget w;
```

```
Screen *XtScreen(w)
    Widget w;
```

```
Window XtWindow(w)
    Widget w;
```

These macros take a widget and return the specified value.

Several window attributes are locally cached in the widget. Thus, they can be set by the resource manager and XtSetValues, as well as used by routines that derive structures from these values (for example, depth for deriving pixmaps, background_pixel for deriving GCs, and so on) or in the XtCreateWindow call.

The x, y, width, height, and border_width window attributes are available to geometry managers. These fields are maintained synchronously inside the X Toolkit. When an XConfigureWindow is issued on the widget's window (on request of its parent), these values are updated immediately rather than sometime later when the server gets around to generating a ConfigureNotify event. (In fact, most widgets do not have SubstructureNotify turned on.) This ensures that all geometry calculations are based on the internally consistent toolkit world, rather than on either of the following:

- An inconsistent world updated by asynchronous ConfigureNotify events
- A consistent but slow world in which geometry managers ask the server for window sizes whenever they need to layout their managed children. See "Geometry Management" for

further information.

4.5. Creating Widgets

The creation of widget instances is a three-phase process:

1. The widgets are allocated and initialized with resources and are optionally added to the managed subset of their parent.
2. All composite widgets are notified of their managed children in a bottom-up traversal of the widget tree.
3. The widgets create X windows that, then, get mapped.

To start the first phase, the application calls XtCreateWidget for all its widgets and adds some (usually, most or all) of its widgets to their respective parent's managed set by calling XtManageChild. In order to avoid an O(n^2) creation process where each composite widget lays itself out each time a widget is created and managed, parent widgets are not notified of changes in their managed set during this phase.

After all widgets have been created, the application calls XtRealizeWidget on the top-level widget to start the second and third phases. XtRealizeWidget first recursively traverses the widget tree in a post-order (bottom-up) traversal and then notifies each composite widget with one or more managed children by means of its change_managed procedure.

Notifying a parent about its managed set involves geometry layout and possibly geometry negotiation. A parent deals with constraints on its size imposed from above (as when a user specifies the application window size), and suggestions made from below (as when a primitive child computes its preferred size). The clash between the two can cause geometry changes to ripple in both directions through the widget tree. The parent may force some of its children to change size and position and may issue geometry requests to its own parent in order to better accommodate all its children. You do not really know where anything should go on the screen until this process settles down.

Consequently, in the first and second phases, no X windows are actually created because it is highly likely that they would just get moved around after creation. This avoids unnecessary requests to the X server.

Finally, XtRealizeWidget starts the third phase by making a pre-order (top-down) traversal of the widget tree, and allocates an X window to each widget by means of its realize procedure, and finally maps the widgets that are managed.

4.5.1. Creating and Merging Argument Lists

Many Intrinsics routines need to be passed pairs of resource names and values. These are passed as an ArgList, which contains:

```
typedef long XtArgVal;

typedef struct {
        String name;
        XtArgVal value;
} Arg, *ArgList;
```

If the size of the resource is less than or equal to the size of an XtArgVal the resource value is stored directly in value. Otherwise, a pointer to it is stored into value.

To set values in an ArgList, use XtSetArg.

```
XtSetArg(arg, name, value)
    Arg arg;
    String name;
    XtArgVal value;
```

arg	Specifies the name-value pair to set.
name	Specifies the name of the resource.
value	Specifies the value of the resource if it will fit in an XtArgVal, otherwise the address.

An ArgList usually is specified in a highly stylized manner in order to minimize the probability of making a mistake, for example:

```
Arg args[20];
int n;

n = 0;
XtSetArg(args[n], XtNheight, 100);   n++;
XtSetArg(args[n], XtNwidth, 200);    n++;
XtSetValues(widget, args, n);
```

Note

You should not use auto-increment or auto-decrement within the first argument to XtSetArg. As it is currently implemented, XtSetArg is a macro that dereferences the first argument twice.

To merge two ArgList structures, use XtMergeArgLists.

```
ArgList XtMergeArgLists(args1, num_args1, args2, num_args2)
    ArgList args1;
    Cardinal num_args1;
    ArgList args2;
    Cardinal num_args2;
```

args1	Specifies the first ArgList to include.
num_args1	Specifies number of arguments in the first ArgList.
args2	Specifies the second ArgList to include.
num_args2	Specifies the number of arguments in the second ArgList.

XtMergeArgLists allocates storage large enough to hold the combined ArgList structures and copies them into it. It does not check for duplicate entries. When it is no longer needed, the returned storage should be freed by the client with XtFree.

4.5.2. Creating a Widget Instance

To create an instance of a widget, use XtCreateWidget.

```
Widget XtCreateWidget(name, widget_class, parent, args, num_args)
    String name;
    WidgetClass widget_class;
    Widget parent;
    ArgList args;
    Cardinal num_args;
```

name	Specifies the resource name for the created widget. This name is used for retrieving resources and, for that reason, should not be the same as any other widget that is a child of same parent.
widget_class	Specifies the widget class pointer for the created widget.
parent	Specifies the parent widget.
args	Specifies the argument list to override the resource defaults.
num_args	Specifies the number of arguments in args. The number of arguments in an argument list can be automatically computed by using the XtNumber macro. For further information, see "Determining the Number of Elements".

XtCreateWidget performs much of the "boiler-plate" operations of widget creation. That is, it performs the following:

- Checks to see if class_initialize has been called for this class and for all superclasses and, if not, calls those necessary in a superclass to subclass order.
- Checks that the parent is a subclass of compositeWidgetClass.
- Allocates memory for the widget instance.
- If the parent is a subclass of constraintWidgetClass, it allocates memory for the parent's constraints and stores the address of this memory into the constraints field.
- Initializes the core nonresource data fields (for example, parent and visible).
- Initializes the resource fields (for example, background_pixel) by using the resource lists specified for this class and all superclasses.
- If the parent is a subclass of constraintWidgetClass, it initializes the resource fields of the constraints record by using the constraint resource list specified for the parent's class and all superclasses up to constraintWidgetClass.
- Calls the initialize procedures for the widget, starting at the Core initialize procedure on down to the widget's initialize procedure.
- If the parent is a subclass of constraintWidgetClass, it calls the constraint initialize procedures, starting at constraintWidgetClass on down to the parent's constraint initialize procedure.
- Puts the widget into its parent's children list by calling its parent's insert_child procedure. For further information, see "Addition of Children to a Composite Widget".

4.5.3. Creating an Application Shell Instance

To create an instance of an application shell widget, use XtCreateApplicationShell.

```
Widget XtCreateApplicationShell(name, widget_class, args, num_args)
    String name;
    WidgetClass widget_class;
    ArgList args;
    Cardinal num_args;
```

name	Specifies the resource name for the created application shell widget.
widget_class	Specifies the widget class pointer for the created application shell widget. This will usually be topLevelShellWidgetClass or a subclass thereof.
args	Specifies the argument list to override the resource defaults.
num_args	Specifies the number of arguments in args.

XtCreateApplicationShell creates another top-level widget that is the root of another widget tree. The initial top-level widget is returned from XtInitialize. An application uses this procedure if it needs to have several independent windows.

4.5.4. Initialization of a Widget Instance: the initialize procedure

The initialize procedure for a widget class is of type XtInitProc:

```
typedef void (*XtInitProc)();

void InitProc(request, new)
    Widget request, new;
```

request	Specifies the widget with resource values as requested by the argument list, the resource database, and the widget defaults.
new	Specifies a widget with the new values, both resource and non-resource, that are actually allowed.

The three main jobs of an initialization procedure are to:

- Allocate space for and copy any resources that are referenced by address. For example, if a widget has a field that is a string (char *) it cannot depend upon the characters at that address remaining constant but must dynamically allocate space for the string and copy it to the new space. (Note that you should not allocate space for or copy callback lists.)
- Compute values for unspecified resource fields. For example, if width and height are zero (0), the widget should compute a nice width and height based on other resources. This is the only time that a widget may ever directly assign its own width and height.
- Compute values for uninitialized non-resource fields that are derived from resource fields. For example, GCs that the widget uses are derived from resources like background, foreground, and font.

An initialization procedure can also check certain fields for internal consistency. For example, it makes no sense to specify a color map for a depth that does not support that color map.

Initialization procedures are called in ''superclass-to-subclass order''. Most of the initialization code for a specific widget class deals with fields defined in that class and not with fields defined in its superclasses.

If a subclass does not need an initialization procedure because it does not need to perform any of the above operations, you can specify NULL for the initialize field in the class record.

Sometimes a subclass may want to overwrite values filled in by its superclass. In particular, size calculations of a superclass are often incorrect for a subclass and, in this case, the subclass must modify or recalculate fields declared and computed by its superclass.

As an example, a subclass can visually surround its superclass display. In this case, the width and height calculated by the superclass initialize procedure are too small and need to be incremented by the size of the surround. The subclass needs to know if its superclass's size was calculated by the superclass or was specified explicitly. All widgets must place themselves into whatever size is explicitly given, but they should compute a reasonable size if no size is requested. How does a subclass know the difference between a specified size and a size computed by a superclass?

The request and new parameters provide the necessary information. The ''request'' widget is the widget as originally requested. The ''new'' widget starts with the values in the request, but it has been updated by all superclass initialization procedures called so far. A subclass initialize procedure can compare these two to resolve any potential conflicts.

In the above example, the subclass with the visual surround can see if the width and height in the request widget are zero. If so, it adds its surround size to the width and height fields in the new widget. If not, it must make do with the size originally specified.

The ''new'' widget will become the actual widget instance record. Therefore, if the initialization procedure needs to call any routines that operate on a widget, it should specify ''new'' as the widget instance.

4.5.5. Initialization of a Constraint Widget Instance: the constraint_initialize procedure

The constraint initialize procedure is of type XtInitProc. The values passed to the parent constraint initialization procedure are the same as those passed to the child's class widget initialization procedure.

The constraint initialization procedure should compute any constraint fields derived from constraint resources. It can make further changes to the widget in order to make the widget conform to the specified constraints, changing, for example, the widget's size or position.

If a constraint class does not need a constraint initialization procedure, it should specify NULL for the initialize field of the ConstraintClassPart in the class record.

4.5.6. Initialization of Nonwidget Data: the initialize_hook procedure

The initialize_hook procedure is of type XtArgsProc:

```
typedef void (*XtArgsProc)();

void ArgsProc(w, args, num_args)
    Widget w;
    ArgList args;
    Cardinal *num_args;
```

w	Specifies the widget.
args	Specifies the argument list to override the resource defaults.
num_args	Specifies the number of arguments in args.

If this procedure is not NULL, it is called immediately after the corresponding initialize procedure, or in its place if the initialize procedure is NULL.

The initialize_hook procedure allows a widget instance to initialize non-widget data using information from the arglist. For example, the Text widget has subparts that are not widgets, yet these subparts have resources that can be specified by means of the resource file or an argument list. See also ''XtGetSubresources''.

4.6. Realizing Widgets

To realize a widget instance, use XtRealizeWidget.

```
void XtRealizeWidget(w)
    Widget w;
```

w	Specifies the widget.

XtRealizeWidget performs the following:

- If the widget is already realized, XtRealizeWidget simply returns.
- Otherwise, it makes a post-order traversal of the widget tree rooted at the specified widget and calls the change_managed procedure of each composite widget that has one or more managed children.
- It then constructs an XSetWindowAttributes structure filled in with information derived from the Core widget fields and calls the realize procedure for the widget, which adds any widget-specific attributes and creates the X window.
- If the widget is a primitive widget, nothing else need be done, and XtRealizeWidget returns. Otherwise, it recursively descends to each of the widget's managed children and calls the realize procedures.
- Finally, XtRealizeWidget maps all of the managed children windows that have mapped_when_managed TRUE. (If a widget is managed, but mapped_when_managed is FALSE, the widget is allocated visual space but is not displayed. Some people seem to like this to indicate certain states.)

If num_children equals num_mapped_children, XtRealizeWidget calls XMapSubwindows to map all the children at once. Otherwise, it maps each child individually. If the widget is a top-level shell widget (that is, it has no parent), XtRealizeWidget maps the widget window.

XtCreateWidget, XtRealizeWidget, XtManageChildren, XtUnmanageChildren, and XtDestroyWidget maintain the following invariants:

- If w is realized, then all managed children of w are realized.
- If w is realized, then all managed children of w that are also mapped_when_managed are mapped.

All Intrinsic routines and all widget routines should work with either realized or unrealized widgets.

To check whether or not a widget has been realized, use XtIsRealized.

```
Boolean XtIsRealized(w)
    Widget w;
```

w	Specifies the widget.

XtIsRealized returns TRUE if the widget has been realized. That is, it returns TRUE if the widget has a nonzero X window ID.

Some widget procedures (for example, set_values) might wish to operate differently after the widget has been realized.

4.6.1. Creation of a Window for a Widget Instance: the realize procedure

The realize procedure for a widget class is of type XtRealizeProc:

```
typedef void (*XtRealizeProc)();

void RealizeProc(w, value_mask, attributes)
    Widget w;
    XtValueMask *value_mask;
    XSetWindowAttributes *attributes;
```

w	Specifies the widget.
value_mask	Specifies which fields in the attributes structure to use.

attributes	Specifies the window attributes to use in the XCreateWindow call.

The realize procedure must make the window a reality.

The generic XtRealizeWidget function fills in a mask and a corresponding XSetWindowAttributes structure. It sets the following fields based on information in the widget Core structure:

- background_pixmap (or background_pixel if background_pixmap is NULL) is filled in from the corresponding field.
- border_pixmap (or border_pixel if border_pixmap is NULL) is filled in from the corresponding field.
- event_mask is filled in based on the event handlers registered, the event translations specified, whether expose is non-NULL, and whether visible_interest is TRUE.
- bit_gravity is set to NorthWestGravity if the expose field is NULL.
- do_not_propagate_mask is set to propagate all pointer and keyboard events up the window tree. A composite widget can implement functionality caused by an event anywhere inside it (including on top of children widgets) as long as children do not specify a translation for the event.

All other fields in attributes (and the corresponding bits in value_mask) can be set by the realize procedure.

A widget class can inherit its realize procedure from its superclass during class initialization. The realize procedure defined for Core simply calls XtCreateWindow with the passed value_mask and attributes, and with windowClass and visual set to CopyFromParent. Both CompositeWidgetClass and ConstraintWidgetClass inherit this realize procedure, and most new widget subclasses can do the same. See ''Inheriting Superclass Operations'' for further information.

The most common noninherited realize procedures set bit_gravity in the mask and attributes to the appropriate value and then create the window. For example, Label sets bit_gravity to WestGravity, CenterGravity, or EastGravity. Consequently, shrinking a Label just moves the bits appropriately, and no Expose event is needed for repainting.

If a composite widget wants to have its children realized in a particular order (typically to control the stacking order) it should call XtRealizeWidget on its children itself in the appropriate order from within its own realize procedure.

4.6.2. Create Window Convenience Routine

Rather than call the Xlib XCreateWindow function explicitly, a realize procedure should call the X Toolkit analog XtCreateWindow. This routine simplifies the creation of windows for widgets.

```
void XtCreateWindow(w, window_class, visual, value_mask, attributes)
    Widget w;
    unsigned int window_class;
    Visual *visual;
    XtValueMask value_mask;
    XSetWindowAttributes *attributes;
```

w	Specifies the widget used to set x, y, and so on
window_class	Specifies the Xlib window class (for example, InputOutput, InputOnly, or CopyFromParent).
visual	Specifies the visual type (usually CopyFromParent).
value_mask	Specifies which fields in attributes to use.

attributes Specifies the window attributes to use in the XCreateWindow call.

XtCreateWindow calls XCreateWindow with values from the widget structure and the passed parameters. Then, it assigns the created window into the widget's window field.
XtCreateWindow evaluates the following fields of the Core widget structure:

- depth
- screen
- parent -> core.window
- x
- y
- width
- height
- border_width

4.7. Destroying Widgets

Destroying widgets is simple. The X Toolkit provides support to:

- Destroy all the children of the widget being destroyed.
- Remove (and unmap) the widget from its parent.
- Call procedures that have been registered to trigger when the widget is destroyed.
- Minimize the number of things a widget has to deallocate when destroyed.
- Minimize the number of XDestroyWindow alls.

To destroy a widget instance, use XtDestroyWidget.

```
void XtDestroyWidget(w)
    Widget w;
```

w Specifies the widget.

XtDestroyWidget provides the only method of destroying a widget, including widgets that wish to destroy themselves. It can be called at any time, including from an application callback routine of the widget being destroyed. This requires a two-phase destroy process in order to avoid dangling references to destroyed widgets.
In phase one, XtDestroyWidget performs the following actions:

- If the being_destroyed field of the widget is TRUE, XtDestroyWidget returns immediately.
- Removes the widget from its parent's managed set which, in turn, causes the widget to be unmapped.
- Sets the being_destroyed field to TRUE and the visible bit to FALSE for the widget and all descendants.
- Adds the widget to a list of widgets (the destroy list) that should be destroyed when it is safe to do so.

Entries on the destroy list satisfy the invariant:

- If w1 occurs before w2 on the destroy list, then there is no ancestor/child relationship between the two, or w1 is a descendant of w2. (The terms "child" and "descendant" here refer to both normal and pop-up children.)

Phase two occurs when all procedures that should execute as a result of the current event have been called (including all procedures registered with the Event and Translation Managers). That is, phase two occurs when XtNextEvent is called.

Issues

1. XtDestroyWidget may get rid of all widgets, and then the next call to XtNextEvent won't ever get any events. So we expect to move phase two to happen at the end of XtDispatchEvent, allowing customized event loops to test a flag before looping back to XtNextEvent.
2. The phase two destroy should happen only at the end of the outermost call to XtDispatchEvent, because there may be nested calls to an event dispatch loop in applications/widgets that maintain some state in the program counter.

In phase two, XtDestroyWidget performs the following actions on each entry in the destroy list:

- Calls the destroy callbacks registered on the widget (and all descendants) in post-order. That is, it calls children callbacks before parent callbacks.
- Calls the widget's parent's delete_child procedure. (See "Deletion of Children".)
- If the widget's parent is a subclass of constraintWidgetClass, it calls the constraint destroy procedure for the parent, then the parent's superclass, until finally it calls the constraint destroy procedure for constraintWidgetClass.
- Calls the destroy class procedures for the widget (and all descendants) in post-order. For each such widget, it calls the destroy procedure declared in the widget class, then the destroy procedure declared in its superclass, until finally it calls the destroy procedure declared in the Core class record.
- Calls XDestroyWindow if the widget is realized (that is, has an X window). The server recursively destroys all descendant windows.

Finally, XtDestroyWidget recursively descends the tree and deallocates all widgets and constraint records.

4.7.1. Adding and Removing Destroy Callbacks

The destroy callback uses the mechanism described in "Callbacks". The destroy callback list is identified by the resource name XtNdestroyCallback. To add a destroy callback procedure ClientDestroy with client data *client_data* to Widget w, call XtAddCallback. To remove the callback, call XtRemoveCallback. Both calls take the following parameter list:

(*w*, XtNdestroyCallback, *ClientDestroy*, *client_data*)

4.7.2. Deallocation of Dynamic Data: the destroy procedure

The destroy procedure is of type XtWidgetProc:

```
typedef void (*XtWidgetProc)();

void WidgetProc(w)
    Widget w;
```

w Specifies the widget.

The destroy procedures are called in subclass-to-superclass order. Therefore, a widget's destroy procedure should only deallocate storage that is specific to the subclass and should not bother with the storage allocated by any of its superclasses. If a widget does not need to deallocate any storage, the destroy procedure entry in its widget class record can be NULL.

Deallocating storage includes but is not limited to:

- Calling XtFree on dynamic storage allocated with XtMalloc, XtCalloc, and so on.
- Calling XtRemoveAllCallbacks on callback lists.
- Calling XtDestroyPixmap on pixmaps allocated with XtGetPixmap.
- Calling XFreePixmap on pixmaps created with direct X calls.
- Calling XtDestroyGC on GCs allocated with XtGetGC.
- Calling XFreeGC on GCs allocated with direct X calls.
- Calling XtRemoveEventHandler on event handlers added with XtAddEventHandler.
- Calling XtRemoveTimeOut on timers created with XtAddTimeOut.

4.7.3. Deallocation of Dynamic Constraint Data: the constraint destroy procedure

The constraint destroy procedure is of type XtWidgetProc. They are called for a widget whose parent is a subclass of constraintWidgetClass. The constraint destroy procedures are called in subclass-to-superclass order, starting at the widget's parent and ending at constraintWidgetClass. Therefore, a parent's constraint destroy procedure should only deallocate storage that is specific to the constraint subclass and not the storage allocated by any of its superclasses.

If a parent does not need to deallocate any constraint storage, the constraint destroy procedure entry in its class record can be NULL.

4.8. Exiting an Application

All X Toolkit applications that wish to terminate should just do so by calling XCloseDisplay and exiting using the standard method for their operating system (typically, by calling exit for UNIX-based systems). The quickest way to make the windows disappear while exiting is to call XtUnmapWidget on each top-level shell widgets. The X Toolkit has no resources beyond those in the program image, and the X server will free its resources when its connection to the application is broken.

Chapter 5

Callbacks

Applications and other widgets (clients) often want to register a procedure with a widget that gets called under certain conditions. For example, when a widget is destroyed, every procedure on the widget's destroy_callbacks list is called to notify clients of the widget's impending doom.

Every widget has a destroy_callbacks list. Widgets can define additional callback lists as they see fit. For example, the Command widget has a callback list to notify clients when the button has been activated.

5.1. Callback Procedure and Callback List Definitions

Callback procedures for use in callback lists are of type XtCallbackProc:

```
typedef void (*XtCallbackProc)();

void CallbackProc(w, client_data, call_data)
     Widget w;
     caddr_t client_data;
     caddr_t call_data;
```

w	Specifies widget for which the callback is registered.
client_data	Specifies the data that the widget should pass back to the client when the widget executes the client's callback procedure. This is a way for the client registering the callback to also register client-specific data: a pointer to additional information about the widget, a reason for invoking the callback, and so on. It is perfectly normal to have client_data be NULL if all necessary information is in the widget.
call_data	Specifies any callback-specific data the widget wants to pass to the client. For example, when Scrollbar executes its thumbChanged callback list, it passes the new position of the thumb. The call_data argument merely is a convenience to avoid having simple cases where the client could otherwise need to call XtGetValues or a widget-specific function to retrieve data from the widget. Complex state information in call_data generally should be avoided. The client can use the more general data retrieval methods, if necessary.

Whenever a client wants to pass a callback list as an argument in an XtCreateWidget, XtSetValues, or XtGetValues call, it should specify the address of a NULL-terminated array of type XtCallbackList:

```
typedef struct {
        XtCallbackProc callback;
        caddr_t client_data;
} XtCallbackRec, *XtCallbackList;
```

For example, the callback list for procedures A and B with client data clientDataA and clientDataB, respectively, is:

```
static XtCallbackRec callbacks[] = {
        {A, (caddr_t) clientDataA},
        {B, (caddr_t) clientDataB},
        {(XtCallbackProc) NULL, (caddr_t) NULL}
};
```

Though callback lists are passed by address in argument lists, the Intrinsics know about callback lists. Your widget initialize and set_values procedures should not allocate memory for the callback list. The Intrinsics do this for you, using a different structure for their internal representation.

5.2. Identifying Callback Lists

Whenever a widget contains a callback list for use by clients, it also exports in its public ".h" file the resource name of the callback list. Applications and client widgets never access callback list fields directly. Instead, they always identify the desired callback list using the exported resource name. All callback manipulation routines described below check that the requested callback list is indeed implemented by the widget.

In order for the Intrinsics to find and correctly handle callback lists, they should always be declared with a resource type of XtRCallback.

5.3. Adding Callback Procedures

To add a callback procedure to a callback list, use XtAddCallback.

```
void XtAddCallback(w, callback_name, callback, client_data)
    Widget w;
    String callback_name;
    XtCallbackProc callback;
    caddr_t client_data;
```

w	Specifies the widget to add the callback to.
callback_name	Specifies the callback list within the widget to append to.
callback	Specifies the callback procedure to add.
client_data	Specifies the client data to be passed to the callback when it is invoked by XtCallCallbacks. The client_data parameter is often NULL).

To add a list of callback procedures to a callback list, use XtAddCallbacks.

```
void XtAddCallbacks(w, callback_name, callbacks)
    Widget w;
    String callback_name;
    XtCallbackList callbacks;
```

w	Specifies the widget to add the callbacks to.
callback_name	Specifies the callback list within the widget to append to.
callbacks	Specifies the null-terminated list of callback procedures and corresponding client data to add.

5.4. Removing Callback Procedures

To remove a callback procedure from a callback list, use XtRemoveCallback.

```
void XtRemoveCallback(w, callback_name, callback, client_data)
    Widget w;
    String callback_name;
    XtCallbackProc callback;
    caddr_t client_data;
```

w	Specifies the widget to delete the callback from.
callback_name	Specifies the callback list within the widget to remove the callback from.
callback	Specifies the callback procedure to delete.
client_data	Specifies the client data to match on the registered callback procedure. (The XtRemoveCallback routine removes a callback only if both the procedure and the client data match).

To remove a list of callback procedures from a callback list, use XtRemoveCallbacks.

```
void XtRemoveCallbacks(w, callback_name, callbacks)
    Widget w;
    String callback_name;
    XtCallbackList callbacks;
```

w	Specifies the widget to delete the callbacks from.
callback_name	Specifies the callback list within the widget to remove the callbacks from.
callbacks	Specifies the list of callbacks to delete.

To remove all callback procedures from a callback list (and, thus, free all storage associated with the callback list), use XtRemoveAllCallbacks.

```
void XtRemoveAllCallbacks(w, callback_name)
    Widget w;
    String callback_name;
```

w	Specifies the widget to remove the callback from.
callback_name	Specifies the callback list within the widget to remove.

5.5. Executing Callback Procedures

To execute the procedures in a callback list, use XtCallCallbacks.

```
void XtCallCallbacks(w, callback_name, call_data)
    Widget w;
    String callback_name;
    caddr_t call_data;
```

w	Specifies the widget containing the callback list that is to be executed.
callback_name	Specifies the callback list within the widget to execute.
call_data	Specifies a callback-list specific data value to pass to each of the callback procedure in the list. The call_data is NULL if no data is needed (for example, the commandActivated callback list in Command needs only to notify its clients that the button has been activated). The call_data is the actual data if only one (32-

bit) long word is needed. The call_data is the address of the data if more than one word is needed.

5.6. Checking the Status of a Callback List

To find out the status of a callback list, use XtHasCallbacks.

```
typedef enum {XtCallbackNoList, XtCallbackHasNone, XtCallbackHasSome} XtCallbackStatus;

XtCallbackStatus XtHasCallbacks(w, callback_name)
    Widget w;
    String callback_name;
```

w	Specifies the widget to check.
callback_name	Specifies the callback list within the widget to check.

XtHasCallbacks first checks if the widget has a callback list identified by callback_name. If not, it returns XtCallbackNoList. Otherwise, it returns XtCallbackHasNone if the callback list is empty, and XtCallbackHasSome if the callback list has at least one callback registered.

Chapter 6

Composite Widgets

Composite widgets can have children. Consequently, they are responsible for much more than primitive widgets. Their responsibilities (either implemented directly by the widget class or indirectly by Intrinsic procedures) include:

- Overall management of children from creation to destruction.
- Destruction of descendants when the composite widget is destroyed.
- Physical arrangement (geometry management) of a displayable subset of children (that is, the "managed" children).
- Mapping and unmapping of a subset of the managed children.
- Focus management for the displayable children.

Overall management is handled by the generic procedures XtCreateWidget and XtDestroyWidget. XtCreateWidget adds children to their parent by calling the parent's insert_child procedure. XtDestroyWidget removes children from their parent by calling the parent's delete_child procedure and ensures all children of a destroyed composite widget also get destroyed.

Only a subset of the total number of children are actually managed by the geometry manager and, hence, possibly visible. For example, a multi-buffer composite editor widget might allocate one child widget per file buffer, but it might only display a small number of the existing buffers. Windows that are in this displayable subset are called "managed" windows and enter into geometry manager calculations. The other children are "unmanaged" windows and, by definition, are not mapped.

Children are added to and removed from the managed set by using XtManageChild, XtManageChildren, XtUnmanageChild, and XtUnmanageChildren, which notify the parent to recalculate the physical layout of its children by calling the parent's change_managed procedure. A convenience routine, XtCreateManagedWidget, calls XtCreateWidget and XtManageChild on the result. It has the same parameters as XtCreateWidget.

Most managed children are mapped, but some widgets can be in a state where they take up physical space but do not show anything. Managed widgets will not be mapped automatically if their map_when_managed field is FALSE. This field default is TRUE and is changed by using XtSetMappedWhenManaged.

Each composite widget class has a geometry manager, which is responsible for figuring out where the managed children should appear within the composite widget's window. Geometry management techniques fall into four classes:

- Fixed boxes have a fixed number of children that are created by the parent. All of these children are managed, and none ever make geometry manager requests.
- Homogeneous boxes treat all children equally and apply the same geometry constraints to each child. Many clients insert and delete widgets freely.
- Heterogeneous boxes have a specific location where each child is placed. This location is usually not specified in pixels, because the window may be resized but is expressed rather in terms of the relationship between a child and the parent or between the child and other specific children. Heterogeneous boxes are usually subclasses of Constraint.
- Shell boxes have only one child, which is exactly the size of the shell. The geometry manager must communicate with the window manager if it exists, and the box must also accept ConfigureNotify events when the window size is changed by the window manager.

Each composite widget, especially those that are heterogeneous, can define ways for one child to change focus to another child by means of the move_focus_to_next and move_focus_to_prev procedures. For example, typing carriage return in one child widget may move to the "next" child widget, while typing a number in one child widget may move focus to any of a number of children widgets.

6.1. Verifying the Class of a Composite Widget

To test if a widget is a subclass of Composite, use XtIsComposite.

```
void XtIsComposite(w)
    Widget w;
```

w Specifies the widget under question.

XtIsComposite (w) is just an abbreviation for XtIsSubclass (*w*, *compositeWidgetClass*).

6.2. Addition of Children to a Composite Widget: the insert_child procedure

To add the child to the parent's children array, XtCreateWidget calls the parent's class routine insert_child. The insert_child procedure for a composite widget is of type XtWidgetProc. An insert_child procedure takes the widget to insert and the argument list used to create the widget.

Most composite widgets just inherit their superclass's operation. Composite's insert_child routine merely calls the insert_position procedure and inserts the child at the specified position.

Some composite widgets define their own insert_child routine so that they can order their children in some convenient way, so that they can create companion "controller" widgets for a new widget, or so they can limit the number or type of their children widgets.

If there is not enough room to insert a new child in the children array (num_children = num_slots), the insert_child procedure must first realloc the array and update num_slots. The insert_child procedure then places the child wherever it wants and increments the num_children field.

6.2.1. Insertion Order of Children: the insert_position procedure

Instances of composite widgets may care about the order in which their children are kept. For example, an application may want a set of command buttons in some logical order grouped by function, while it may want buttons that represent file names to be kept in alphabetical order.

The insert_position procedure for a composite widget instance is of type XtOrderProc:

```
typedef Cardinal (*XtOrderProc)();

Cardinal OrderProc(w)
    Widget w;
```

w Specifies the widget.

Composite widgets that allow clients to order their children (usually homogeneous boxes) can call their widget instance's insert_position function from the class's insert_child procedure to determine where a new child should go in its children array. Thus, a client of a composite class can apply different sorting criteria to widget instances of the class passing in a different insert_position procedure when it creates each composite widget instance.

The return value of the insert_position procedure indicates how many children should go before the widget. Returning zero (0) means before all other children, while returning num_children means after all other children. The default insert_position function returns num_children. This can be overridden by a specific composite widget's resource list or by the argument list provided when the composite widget is created.

6.3. Deletion of Children: the delete_child procedure

XtDestroyWidget eventually causes a call to the parent's class routine delete_child in order to remove the child from the parent's children array.

A deletion procedure is of type XtWidgetProc, and it takes the widget being deleted.

Most widgets just inherit delete_children from their superclass. Composite widgets that create companion widgets define their own delete_children routine to remove these companion widgets.

6.4. Adding and Removing Children from the Managed Set

The X Toolkit provides a set of generic routines to permit the addition of widgets to or the removal of widgets from a composite widget's managed set. These generic routines eventually call the widget's class procedure, change_managed, which is of type XtWidgetProc.

6.4.1. Managing Children

To add a list of widgets to the geometry-managed (and, hence, displayable) subset of their parent widget, the application must first create the widgets by using XtCreateWidget and then call XtManageChildren.

```
typedef Widget *WidgetList;

void XtManageChildren(children, num_children)
     WidgetList children;
     Cardinal num_children;
```

children	Specifies a list of children to add.
num_children	Specifies the number of children to add.

XtManageChildren performs the following:

- Issues an error if the children do not all have the same parent.
- Returns immediately if the common parent is being destroyed.
- Otherwise, for each unique child on the list:
  - The child is ignored if it is already managed or being destroyed.
  - Otherwise, the child is marked as managed, and the parent's num_mapped_children field is incremented if the child has map_when_managed TRUE.
- If the parent is realized and after all children have been marked, XtManageChildren makes some of the newly managed children visible:
  - Calls the change_managed routine of the widgets' parent.
  - Calls XtRealizeWidget on each previously unmanaged child that is unrealized.
  - Maps each previously unmanaged child that has map_when_managed TRUE.

Managing children is independent of the ordering of children and independent of creating and deleting children. The layout routine of the parent should only bother with children whose managed field is TRUE and should ignore all other children. (Note that some composite widgets, especially fixed boxes, call XtManageChild from their insert_child procedure.)

If the parent widget is realized, its change_managed procedure is called to notify the that that its set of managed children has changed. The parent can reposition and resize any of its children. It moves each child as needed by calling the XtMoveWidget procedure. XtMoveWidget first updates the x and y fields and then calls XMoveWindow if the widget is realized.

If the composite widget wishes to change the size or border width of any of its children, it calls the XtResizeWidget procedure. XtResizeWidget first updates the Core fields and then calls XConfigureWindow if the widget is realized.

To add a single child to the managed children of its parent widget, the application must first create the widget by using XtCreateWidget and then call XtManageChild.

```
void XtManageChild(child)
    Widget child;
```

child	Specifies the child to add.

XtManageChild constructs a WidgetList of length one (1) and calls XtManageChildren.

To create and manage a widget in a single procedure, use XtCreateManagedWidget.

```
Widget XtCreateManagedWidget(name, widget_class, parent, args, num_args)
    String name;
    WidgetClass widget_class;
    Widget parent;
    ArgList args;
    Cardinal num_args;
```

name	Specifies the text name for the created widget.
widget_class	Specifies the widget class pointer for the created widget.
parent	Specifies the parent widget.
args	Specifies the argument list to override the resource defaults.
num_args	Specifies the number of arguments in args.

XtCreateManagedWidget is a convenience routine that calls XtCreateWidget followed by XtManageChild.

6.4.2. Unmanaging Children

To remove a list of children from the managed list of their parent, use XtUnmanageChildren.

```
void XtUnmanageChildren(children, num_children)
    WidgetList children;
    Cardinal num_children;
```

children	Specifies the children to remove.
num_children	Specifies the number of children to remove.

XtUnmanageChildren performs the following:

- Issues an error if the children do not all have the same parent.
- Returns immediately if the common parent is being destroyed.
- Otherwise, for each unique child on the list:
  - The child is ignored if it is already unmanaged or being destroyed.
  - Otherwise, XtUnmanagedChildren marks the child as unmanaged.
  - If the child is realized, XtUnmanageChildren makes it non-visible by unmapping it.

  - Decrements the parent's num_mapped_children field if the widget has map_when_managed TRUE.
- Calls the change_managed routine of the widgets' parent after all children have been marked if the parent is realized.

XtUnmanageChildren does not destroy the children widgets. Removing widgets from a parent's managed set is often a temporary banishment, and, some time later, you may add the children again. To destroy widgets entirely, see ''Destroying Widgets''.

To remove a single child from its parent's managed set, use XtUnmanageChild.

```
void XtUnmanageChild(child)
    Widget child;
```

child Specifies the child to remove.

XtUnmanageChild constructs a widget list of length one and calls XtUnmanageChildren.

These generic routines are low-level routines used by ''generic'' composite widget building routines. In addition, composite widgets can provide widget-specific, high-level convenience routines to allow applications to create and manage children more easily.

6.5. Controlling When Widgets Get Mapped

A widget is normally mapped if it is managed. However, this behavior can be overridden by setting the XtNmappedWhenManaged resource for the widget when it is created or by setting the map_when_managed field to False.

To change the map_when_managed field, use XtSetMappedWhenManaged.

```
void XtSetMappedWhenManaged(w, map_when_managed)
    Widget w;
    Boolean map_when_managed;
```

w Specifies the widget.

map_when_managed
Specifies the new value (either True or False).

If the widget is realized and managed and if the new value of map_when_managed is True, XtSetMappedWhenManaged maps the window. If the widget is realized and managed and if the new value of map_when_managed is False, it unmaps the window.

When a widget's mapped_when_managed field is False, the client is responsible for mapping and unmapping the widget.

XtSetMappedWhenManaged is a convenience function that is equivalent to (but slightly faster than) calling XtSetValues and setting the new value for the mappedWhenManaged resource.

To map a widget explicitly, use XtMapWidget.

```
XtMapWidget(w)
    Widget w;
```

w Specifies the widget.

To unmap a widget explicitly, use XtUnmapWidget.

```
XtUnmapWidget(w)
    Widget w;
```

w Specifies the widget.

6.6. Constrained Composite Widgets

Constraint widgets are a subclass of Composite widgets. The name comes from the fact that they manage the geometry of their children based upon constraints associated with each child. Constraints can be as simple as information such as the maximum width and height the parent will allow the child to occupy, or they can be more complicated information such as how other children should change if this child is moved or resized.

Constraint widgets have all the responsibilities of normal composite widgets, and, in addition, must process and act upon the constraint information associated with each of their children.

In order to make it easy for widgets and the Intrinsics to keep track of the constraints a parent associated with each child, every widget has a constraints field. This field is the address of a parent-specific structure containing constraint information about the child. If a child's parent is not a subclass of constraintWidgetClass, then the child's constraints field is NULL.

Subclasses of a constrained widget can add additional constraint fields to their superclass. To allow this, widget writers should define the constraint records in their private ".h" file using the same conventions as used for widget records. For example, a widget that wished to maintain a maximum width and height for each child might define its constraint record like this:

```
typedef struct {
        Dimension max_width, max_height;
} MaxConstraintPart;

typedef struct {
        MaxConstraintPart max;
} MaxConstraintRecord, *MaxConstraint;
```

A subclass of this widget that also wished to maintain a minimum size would define its constraint record thus:

```
typedef struct {
        Dimension min_width, min_height;
} MinConstraintPart;

typedef struct {
        MaxConstraintPart max;
        MinConstraintPart min;
} MaxMinConstraintRecord, *MaxMinConstraint;
```

Constraints are allocated, initialized, deallocated, and otherwise maintained insofar as possible by the Intrinsics. The constraint class record part has several entries that facilitate this. All entries in ConstraintClassPart are information and procedures that are defined and implemented by the parent, but they are called whenever actions are performed upon the parent's children.

XtCreateWidget uses the constraint_size field to allocate a constraint record when a child is created. The constraint_size field gives the number of bytes occupied by a constraint record.

XtCreateWidget uses the constraint resources to fill in resource fields in the constraint record associated with a child. It then calls the constraint initialize procedure so that a the parent can compute constraint fields that are derived from constraint resources and can possible move or resize the child to conform to the given constraints.

XtGetValues and XtSetValues use the constraint resources to get the values or set the values of constraints associated with a child. XtSetValues then calls the constraint set_values procedures so that a parent can recompute derived constraint fields and move or resize the child as appropriate.

XtDestroyWidget calls the constraint destroy procedure to deallocate any dynamic storage associated with a constraint record. The constraint record itself must not be deallocated by the constraint destroy procedure; XtDestroyWidget does this automatically.

Chapter 7

Pop-up Widgets

There are three kinds of pop-ups:

- Modeless pop-ups
- Modal pop-ups
- Spring-loaded pop-ups

A modeless dialog box, an example of a modeless pop-up, is normally visible to the window manager and looks much like just another application from the user point of view. (The application itself is a special form of a modeless pop-up.)

A modal dialog box, an example of a modal pop-up, is not normally visible to the window manager, and, except for events that occur in the dialog box, it disables user-event processing by the application.

A menu, an example of a spring-loaded pop-up, is not visible to the window manager and, except for events that occur in the menu, disables user-event processing by all applications.

Modal pop-ups and spring-loaded pop-ups are really almost the same thing and should be coded as such. In fact, the same widget (for example, a ButtonBox or Menu) can be used both as a modal pop-up or as a spring-loaded pop-up within the same application. The main difference is that spring-loaded pop-ups are brought up with the pointer and, because of the grab that the pointer button causes, they can require different processing by the Intrinsics. Further, button up takes down a spring-loaded pop-up no matter where the button up occurs.

Any kind of pop-up can, in turn, pop up other widgets. Modal and spring-loaded pop-ups can constrain user events to just the most recent such pop-up or to any of the modal/spring-loaded pop-ups currently mapped.

7.1. Pop-ups and the Widget/Window Hierarchy

The one thing all pop-ups have in common is that they break the widget/window hierarchy. Pop-ups windows are not geometry constrained by their parent widget. Instead, they are window children of "root". This means pop-ups are created and attached to their widget parent differently than from normal widget children.

Because the X Toolkit does not support multiple inheritance, and because you can pop up a widget that belongs to any arbitrary widget (for example, Command, MenuBar, Text, and so on), the CorePart record includes the list of the widget's pop-up children. This means that a primitive widget can be a parent, but this is very different from being a composite widget parent. Think of it more like being a godfather. The pop-up list exists mainly to provide the proper place in the widget hierarchy for the pop-up to get resources and to provide a place for XtDestroyWidget to look for all extant children. A parent of a pop-up widget does not actively manage its pop-up children. In fact, it usually never notices them or operates upon them.

A composite widget can have both normal and pop-up children. A pop-up can be popped up from just about anywhere, not just by its parent. The term "child" always refers to a normal, geometry-managed child.on the children list. The term "pop-up child" always refers to a child on the pop-ups list.

7.2. Creating a Pop-up Shell

In order to pop up some arbitrary widget, it must be the only child of a pop-up widget "shell". This shell is responsible for communication with the X window manager on geometry requests and is responsible for proper handling of the bookkeeping associated with actual pop up and pop down. Pop-up shells never allow more than one child.

This shell is always referred to as the "pop-up shell". The single (normal) child is always referred to as the "pop-up child". Both taken together are referred to as the "pop-up".

To create a pop-up shell, use XtCreatePopupShell.

```
Widget XtCreatePopupShell(name, widget_class, parent, args, num_args)
    String name;
    WidgetClass widget_class;
    Widget parent;
    ArgList args;
    Cardinal num_args;
```

name	Specifies the text name for the created shell widget.
widget_class	Specifies the widget class pointer for the created shell widget.
parent	Specifies the parent widget.
args	Specifies the argument list to override the resource defaults.
num_args	Specifies the number of arguments in args.

XtCreatePopupShell ensures that the specified class is a subclass of Shell and that rather than using insert_child to attach the widget to the parent's children list just attaches the shell to the parent's pop-ups list directly.

To use a spring-loaded pop-up, the pop-up shell must be created at application start-up so that the translation manager can find the shell by name. Otherwise, the pop-up shell can be created "on-the-fly" when the pop-up is actually needed. This delayed creation of the shell is particularly useful when you pop up an unspecified number of pop-ups. You can look to see if an appropriate unused shell (that is, not currently popped up) exists and create a new shell if needed.

7.3. Creating Pop-up Children

Once a pop-up shell is created, the single child of the pop-up shell can be created. The two styles for this are:

- Static
- Dynamic

At application start up, an application can create the child of the pop-up shell. This is appropriate for pop-up children that are composed of a fixed set of widgets, and the application can change the state of the subparts of the pop-up child as the application state changes. For example, if an application creates a static menu, it can call XtSetSensitive (or, in general, XtSetValues) on any of the buttons that make up the menu. Creating the pop-up child early means that pop-up time is minimized, especially if the application calls XtRealizeWidget on the pop-up shell at startup time. When the menu is needed, all the widgets that make up the menu already exist and need only be mapped. The menu should pop up as quickly as the X server can respond.

Alternatively, an application can postpone the creation of the child until it is needed. This minimizes application startup time and allows the pop-up child to reconfigure itself each time it is popped up. In this case, the pop-up child creation routine should "poll" the application to find out if it should change the sensitivity of any of its subparts.

Pop-up child creation does not map the pop-up, even if you create the child and call XtRealizeWidget on the pop-up shell. All pop-up shells automatically perform an XtManageChild on their child within their insert_child procedure. There is no need for the creator of a pop-up child to call XtManageChild.

All shells have pop-up and pop-down callbacks. These provide the opportunity either to make last-minute changes to a pop-up child before it is popped up or to change it after it is popped down. Programmers should be aware that excessive use of pop-up callbacks can make popping up occur more slowly.

7.4. Mapping a Pop-up Widget

Pop-ups can be popped up through several mechanisms:

- A call to XtPopup.
- One of the supplied callback procedures (for example, XtCallbackNone, XtCallback-Nonexclusive, or XtCallbackExclusive).
- The standard translation action MenuPopup.

Some of these routines take an argument of type XtGrabKind, which is defined as:

```
typedef enum {XtGrabNone, XtGrabNonexclusive, XtGrabExclusive} XtGrabKind;
```

To map a pop-up from within an application, use XtPopup.

```
void XtPopup(popup_shell, grab_kind)
    Widget popup_shell;
    XtGrabKind grab_kind;
```

popup_shell	Specifies the widget shell to pop up.
grab_kind	Specifies the way in which user events should be constrained.

XtPopup performs the following actions:

- Calls XtCheckSubclass to ensure popup_shell is a subclass of Shell.
- Generates an error if the shell's popped_up field is already TRUE.
- Calls the callback procedures on the shell's popup_callback list.
- Sets the shell popped_up field to TRUE, the shell spring_loaded field to FALSE, and the shell grab_kind field from grab_kind.
- If the shell's create_popup_child field is non-NULL, XtPopup calls it with popup_shell as the parameter.
- If grab_kind is either XtGrabNonexclusive or XtGrabExclusive, XtPopup calls:

  ```
  XtAddGrab(popup_shell, (grab_kind == XtGrabExclusive), FALSE)
  ```

- Calls XtRealizeWidget(popup_shell)
- Calls XMapWindow(popup_shell)

To map a pop-up from a callback list, you can use the XtCallbackNone, XtCallbackNonexclusive, or XtCallbackExclusive convenience routines.

```
void XtCallbackNone(w, client_data, call_data)
    Widget w;
    XtClosure client_data;
    caddr_t call_data;
```

w	Specifies the widget executing the callback.

client_data	Specifies the pop-up shell to pop up.
call_data	Specifies the callback data. This parameter is not used by this procedure.

```
void XtCallbackNonexclusive(w, client_data, call_data)
    Widget w;
    XtClosure client_data;
    caddr_t call_data;
```

w	Specifies the widget executing the callback.
client_data	Specifies the pop-up shell to pop up.
call_data	Specifies the callback data. This parameter is not used by this procedure.

```
void XtCallbackExclusive(w, client_data, call_data)
    Widget w;
    XtClosure client_data;
    caddr_t call_data;
```

w	Specifies the widget executing the callback.
client_data	Specifies the pop-up shell to pop up.
call_data	Specifies the callback data. This parameter is not used by this procedure.

Each of these routines calls XtPopup with the shell specified by the client data parameter and grab_kind set as the name specifies. XtCallbackNone specifies XtGrabNone, and so on. Each then sets the widget that executed the callback list to be insensitive.

The use of these routines in callbacks is not required. In particular, callbacks that create pop-up shells dynamically or that must do more than desensitizing the button will have custom code.

To pop up a menu when a pointer button is pressed or when the pointer is moved into some window, use MenuPopup. From a translation writer's point of view, the definition for this translation action is:

```
void MenuPopup(shell_name)
    String shell_name;
```

shell_name	Specifies the name of the widget shell to pop up.

MenuPopup is known to the translation manager, which must perform special actions for spring-loaded pop-ups. Calls to MenuPopup in a translation specification are mapped into calls to a non-exported action procedure and the translation manager fills in parameters based upon the event specified on the left-hand side of a translation.

If MenuPopup is invoked upon ButtonPress (possibly with modifiers), the translation manager pops up the shell with grab_kind XtExclusive and spring_loaded TRUE. If MenuPopup is invoked upon EnterWindow (possibly with modifiers), the translation manager pops up the shell with grab_kind XtNonexclusive and spring_loaded FALSE. Otherwise, the translation manager generates an error. When the widget is popped up, the following actions are performed:

- Calls XtCheckSubclass to ensure popup_shell is a subclass of Shell.
- Generates an error if the shell's popped_up field is already TRUE.

- Calls the callback procedures on the shell's popup_callback list.
- Sets the shell popped_up field to TRUE and the shell grab_kind and spring_loaded fields appropriately.
- If the shell's create_popup_child field is non-NULL, it is called with popup_shell as the parameter.
- MenuPopup then calls:

  XtAddGrab(popup_shell, (grab_kind == XtGrabExclusive), spring_loaded)

- Calls XtRealizeWidget(popup_shell)
- Calls XMapWindow(popup_shell)

Note that these actions are the same as those for XtPopup.

MenuPopup tries to find the shell by looking up the widget tree starting at the parent of the widget in which it is invoked. If it finds a shell with the specified name in the pop-up children of that parent, it pops up the shell with the appropriate parameters. Otherwise, it moves up the parent chain as needed. If MenuPopup gets to the application widget and cannot find a matching shell, it generates an error.

7.5. Unmapping a Pop-up Widget

Pop-ups can be popped down through several mechanism:

- A call to XtPopdown.
- The supplied callback procedure XtCallbackPopdown.
- The standard translation action MenuPopdown.

To unmap a pop-up from within an application, use XtPopdown.

```
void XtPopdown(popup_shell)
    Widget popup_shell;
```

popup_shell Specifies the widget shell to pop down.

XtPopdown performs the following:

- Calls XtCheckSubclass to ensure popup_shell is a subclass of Shell.
- Checks that popup_shell is currently popped_up. If not, it generates an error.
- Unmaps popup_shell's window.
- If popup_shell's grab_kind is either XtGrabNonexclusive or XtGrabExclusive, calls XtRemoveGrab.
- Sets popup_shell's popped_up field to FALSE.
- Calls the callback procedures on the shell's popdown_callback list.

Pop-ups that have been popped up with one of the callback routines (that is, XtCallbackNone, XtCallbackNonexclusive, XtCallbackExclusive) can be popped down by the callback routine XtCallbackPopdown.

```
void XtCallbackPopdown(w, client_data, call_data)
    Widget w;
    XtClosure client_data;
    caddr_t call_data;
```

w	Specifies the widget executing the callback.
client_data	Specifies the pop-up shell to pop down and the widget used to originally pop it up.
call_data	Specifies the callback data. This parameter is not used by this procedure.

XtCallbackPopdown casts the client data parameter to an XtPopdownID pointer:

```
typedef struct {
        Widget shell_widget;
        Widget enable_widget;
} XtPopdownIDRec, *XtPopdownID;
```

XtCallbackPopdown calls XtPopdown with the specified shell_widget. It then calls XtSet-Sensitive to resensitize the enable_widget.

To pop down a spring-loaded menu when a pointer button is released or when the pointer is moved into some window, use MenuPopdown. From a translation writer's point of view, the definition for this translation action is:

```
void MenuPopdown()
```

MenuPopdown calls XtPopdown with the widget for which the translation is specified.

Chapter 8

Shell Widgets

Shell widgets hold an application's top-level widgets to allow them to communicate with the window manager. Shells have been designed to be as nearly invisible as possible. That is, while clients have to create them, they should never have to worry about their sizes.

If a shell widget is resized from the outside, typically by a window manager, the shell widget will resize its child widget automatically. Similarly, if the shell's child widget wants to change size, it can make a geometry request to the shell, and the shell will negotiate the size change with the outer environment. Clients should never attempt to change the size of their shells directly.

There are three classes of public shells:

OverrideShell	This class is used for shell windows that completely bypass the window manager. Pop-up menu shells will typically be of this class or a subclass.
TransientShell	This class is used for shell windows that can be manipulated by the window manager but are not allowed to be separately iconified. They get iconified by the window manager if the main application shell is iconified. Dialog boxes that make no sense without their associated application will typically be in a shell of this class or a subclass.
TopLevelShell	This class is used for normal top level windows. Any additional top-level widgets an application needs will typically be in a shell of this class or a subclass.

8.1. Shell Widget Definitions

Widgets negotiate their size and position with their parent widget, the widget that directly contains them. Widgets at the top level of the hierarchy do not have parent widgets; instead they must deal with the outside world. To provide for this, top level widgets are encapsulated in special widgets called ''Shells''.

Shell widgets are subclasses of the Composite widget. They encapsulate other widgets and can allow a widget to ''jump out'' of the geometry clipping imposed by the parent/child window relationship. If desired, they provide a layer of communication with the window manager.

There are six different types of shell:

Shell	This is the base class for shell widgets and provides fields needed for all types of shells. Shell is a direct subclass of Composite.
OverrideShell	This class is used for shell windows that completely bypass the window manager. It is a subclass of Shell.
WMShell	Contains fields needed by the common window manager protocol. It is a subclass of Shell.
VendorShell	Contains fields used by vendor-specific window managers. It is a subclass of WMShell.

TransientShell	This class is used for shell windows that can be manipulated by the window manager but are not allowed to be iconified. It is a subclass of VendorShell.
TopLevelShell	This class is used for normal top level windows. It is a subclass of VendorShell.

The classes Shell, WMShell, and VendorShell are internal and should not be instantiated. Only OverrrideShell, TransientShell, and TopLevelShell. are for public use.

8.1.1. ShellClassPart Definitions

No shell widget classes have any additional fields:

```
typedef struct { int empty; } ShellClassPart, OverrideShellClassPart,
        WMShellClassPart, VendorShellClassPart, TransientShellClassPart,
        TopLevelShellClassPart;
```

Shell widget classes have the (empty) shell fields immediately following the composite fields:

```
typedef struct _ShellClassRec {
        CoreClassPart core_class;
        CompositeClassPart composite_class;
        ShellClassPart shell_class;
} ShellClassRec;

typedef struct _OverrideShellClassRec {
        CoreClassPart core_class;
        CompositeClassPart composite_class;
        ShellClassPart shell_class;
        OverrideShellClassPart override_shell_class;
} OverrideShellClassRec;

typedef struct _WMShellClassRec {
        CoreClassPart core_class;
        CompositeClassPart composite_class;
        ShellClassPart shell_class;
        WMShellClassPart wm_shell_class;
} WMShellClassRec;

typedef struct _VendorShellClassRec {
        CoreClassPart core_class;
        CompositeClassPart composite_class;
        ShellClassPart shell_class;
        WMShellClassPart wm_shell_class;
        VendorShellClassPart vendor_shell_class;
} VendorShellClassRec;

typedef struct _TransientShellClassRec {
        CoreClassPart core_class;
        CompositeClassPart composite_class;
        ShellClassPart shell_class;
        WMShellClassPart wm_shell_class;
        VendorShellClassPart vendor_shell_class;
        TransientShellClassPart transient_shell_class;
} TransientShellClassRec;

typedef struct _TopLevelShellClassRec {
        CoreClassPart core_class;
        CompositeClassPart composite_class;
        ShellClassPart shell_class;
        WMShellClassPart wm_shell_class;
        VendorShellClassPart vendor_shell_class;
        TopLevelShellClassPart top_level_shell_class;
} TopLevelShellClassRec;
```

The predefined class records and pointers for shells are:

```
extern ShellClassRec shellClassRec;
extern OverrideShellClassRec overrideShellClassRec;
extern WMShellClassRec wmShellClassRec;
extern VendorShellClassRec vendorShellClassRec;
extern TransientShellClassRec transientShellClassRec;
extern TopLevelShellClassRec topLevelShellClassRec;

extern WidgetClass shellWidgetClass;
extern WidgetClass overrideShellWidgetClass;
extern WidgetClass wmShellWidgetClass;
extern WidgetClass vendorShellWidgetClass;
extern WidgetClass transientShellWidgetClass;
extern WidgetClass topLevelShellWidgetClass;
```

The following opaque types and the opaque variables are defined for generic operations on widgets that are a subclass of ShellWidgetClass.

Types	Variables
ShellWidget	shellWidgetClass
OverrideShellWidget	overrideShellWidgetClass
WMShellWidget	wmShellWidgetClass
VendorShellWidget	vendorShellWidgetClass
TransientShellWidget	transientShellWidgetClass
TopLevelShellWidget	topLevelShellWidgetClass
ShellWidgetClass	
OverrideShellWidgetClass	
WMShellWidgetClass	
VendorShellWidgetClass	
TransientShellWidgetClass	
TopLevelShellWidgetClass	

8.1.2. ShellPart Definition

The various shells have the following additional fields defined in their widget records:

```
typedef struct {
        String geometry;
        XtCreatePopupChildProc create_popup_child_proc;
        XtGrabKind grab_kind;
        Boolean spring_loaded;
        Boolean popped_up;
        Boolean allow_shell_resize;
        Boolean client_specified;
        Boolean save_under;
        Boolean override_redirect;
        XtCallbackList popup_callback;
        XtCallbackList popdown_callback;
} ShellPart;

typedef struct { int empty; } OverrideShellPart;

typedef struct {
        String title;
        int wm_timeout;
        Boolean wait_for_wm;
        Boolean transient;
        XSizeHints size_hints;
        XWMHints wm_hints;
} WMShellPart;

typedef struct {
        int vendor_specific;
} VendorShellPart;

typedef struct { int empty; } TransientShellPart;

typedef struct {
        String icon_name;
        Boolean iconic;
} TopLevelShellPart;
```

The full definitions of the various shell widgets have shell fields following composite fields:

```
typedef struct {
        CorePart core;
        CompositePart composite;
        ShellPart shell;
} ShellRec, *ShellWidget;

typedef struct {
        CorePart core;
        CompositePart composite;
        ShellPart shell;
        OverrideShellPart override;
} OverrideShellRec, *OverrideShellWidget;

typedef struct {
        CorePart core;
        CompositePart composite;
        ShellPart shell;
        WMShellPart wm;
} WMShellRec, *WMShellWidget;

typedef struct {
        CorePart core;
        CompositePart composite;
        ShellPart shell;
        WMShellPart wm;
        VendorShellPart vendor;
} VendorShellRec, *VendorShellWidget;

typedef struct {
        CorePart core;
        CompositePart composite;
        ShellPart shell;
        WMShellPart wm;
        VendorShellPart vendor;
        TransientShellPart transient;
} TransientShellRec, *TransientShellWidget;

typedef struct {
        CorePart core;
        CompositePart composite;
        ShellPart shell;
        WMShellPart wm;
        VendorShellPart vendor;
        TopLevelShellPart topLevel;
} TopLevelShellRec, *TopLevelShellWidget;
```

8.1.3. ShellPart Default Values

The default values for all shell fields (filled in by the Shell resource lists and the Shell initialize procedures) are:

Field	Default Value
geometry	NULL
create_popup_child_proc	NULL
grab_kind	(internal)
spring_loaded	(internal)
popped_up	(internal)
allow_shell_resize	FALSE
client_specified	(internal)
save_under	FALSE
override_redirect	TRUE for OverrideShells, FALSE otherwise
popup_callback	NULL

popdown_callback	NULL
title	Icon name, if specified, otherwise the application's name
wm_timeout	5 seconds
wait_for_wm	TRUE
transient	TRUE for TransientShells, FALSE otherwise

The geometry resource can be used to specify size and position. This is usually done only from a command line or a defaults file. The create_popup_child_proc is called by the XtPopup procedure and usually is NULL. The allow_shell_resize field controls whether or not the widget contained by the shell is allowed to try to resize itself. If it is FALSE, any geometry requests will always return XtGeometryNo. Setting save_under instructs the server to attempt to save the contents of windows obscured by the shell when it is mapped and to restore it automatically later. It is useful for pop-up menus. Setting overrideRedirect determines whether or not the shell window will be visible to the window manager. If TRUE, the window will be immediately mapped without the manager's intervention. The popup and popdown callbacks are called during XtPopup and XtPopdown. The title is a string to be displayed by the window manager. The wm_timeout resource limits the amount of time a shell will wait for confirmation of a geometry request to the window manager. If none comes back within that time, the shell decides the window manager is broken and sets wait_for_wm to be FALSE (Later events may reset this value). The wait_for_wm resource sets the initial state for this flag. When the flag is FALSE, the shell does not wait for a response. Rather, it relies upon asynchronous notification. All other resources are for fields in the window manager hints and the window manager size hints. For further information, see *Xlib – C Language X Interface.*

Transient and TopLevel shells all have the following extra resources:

Resource	Default Value
minWidth	none
minHeight	none
maxWidth	none
maxHeight	none
widthInc	none
heightInc	none
minAspectX	none
minAspectY	none
maxAspectX	none
maxAspectY	none
input	FALSE
initialState	Normal
iconPixmap	none
iconWindow	none
iconX	none
iconY	none
IconMask	none
windowGroup	none

TopLevel shells have the the following additional resources:

Field	Default Value
icon_name	Shell widget's name
iconic	FALSE

The icon_name is the string to display in the shell's icon, and iconic is an alternative way to set the initialState resource to indicate that a shell should be initially displayed as an icon.

Chapter 9

Utility Functions

The X Toolkit provides a number of utility functions for:

- Memory management
- Sharing graphics contexts
- Exposure regions
- Error handling

9.1. Memory Management

The X Toolkit memory management routines provide uniform checking for null pointers, and error reporting on memory allocation errors. These routines are completely compatible with the standard C language runtime routines malloc, calloc, realloc, and free with the added functionality:

- XtMalloc, XtNew, XtCalloc, and XtRealloc give an error if there is not enough memory.
- XtFree simply returns if passed a NULL pointer.
- XtRealloc simply allocates new storage if passed a NULL pointer.

See the C library documentation on malloc, calloc, realloc, and free for more information.

To allocate storage, use XtMalloc.

```
char *XtMalloc(size);
    Cardinal size;
```

size Specifies the number of bytes desired.

XtMalloc returns a pointer to a block of storage of at least the specified size bytes. If there is insufficient memory to allocate the new block, XtMalloc calls XtError.

To allocate storage for a new instance of a data type, use XtNew.

```
type *XtNew(type);
    type;
```

type Specifies a previously declared data type

XtNew returns a pointer to the allocated storage. If there is insufficient memory to allocate the new block, XtNew calls XtError. XtNew is an abbreviation for:

```
((type *) XtMalloc((unsigned) sizeof(type))
```

To allocate and initialize an array, use XtCalloc.

```
char *XtCalloc(num, size);
    Cardinal num;
    Cardinal size;
```

num	Specifies the number of array elements to allocate.
size	Specifies the size of an array element in bytes.

XtCalloc allocates space for the specified number of array elements of the specified size bytes and initializes the space to zero. If there is insufficient memory to allocate the new block, XtCalloc calls XtError.

To change the size of an allocated block of storage, use XtRealloc.

```
char *XtRealloc(ptr, num);
    char *ptr;
    Cardinal num;
```

ptr	Specifies a pointer to old storage.
num	Specifies number of bytes desired in new storage.

XtRealloc changes the size of a block of storage (possibly moving it). Then, it copies the old contents (or as much as will fit) into the new block and frees the old block. If there is insufficient memory to allocate the new block, XtRealloc calls XtError. If the specified ptr argument is NULL, XtRealloc allocates the new storage without copying the "old" contents. That is, it simply calls XtMalloc.

To free an allocated block of storage, use XtFree.

```
void XtFree(ptr);
    char *ptr;
```

ptr	Specifies a pointer to the block of storage that is to be freed.

XtFree returns storage and allows it to be reused. If the specified ptr argument is NULL, XtFree returns immediately.

9.2. Sharing Graphics Contexts

The X Toolkit provides a mechanism whereby cooperating clients can share Graphics Contexts, thereby, reducing both the number of Graphics Contexts created and the total number of server calls in any given application. The mechanism implemented is a simple caching scheme and all Graphics Contexts obtained by means of this mechanism must be treated as read-only. If a changeable Graphics Context is needed, the XCreateGC Xlib function should be used instead.

To obtain shared GCs, use XtGetGC.

```
GC XtGetGC(w, value_mask, values)
    Widget w;
    XtGCMask value_mask;
    XGCValues *values;
```

w	Specifies the widget.

value_mask	Specifies which fields of the values are specified. (See XCreateGC.)
values	Specifies the actual values for this GC. (See XCreateGC.)

XtGetGC returns a Graphics Context. The parameters to this function are the same as those for XCreateGC, except that a widget is passed instead of a Display.

XtGetGC shares only GCs in which all values in the GC returned by XCreateGC are the same. In particular, it does not use the value_mask provided to determine which fields of the GC a widget considers relevant. The value_mask is used only to tell the server which fields should be filled in with widget data and which it should fill in with default values.

To deallocate a graphics context when it is no longer needed, use XtDestroyGC.

```
void XtDestroyGC(gc)
    GC gc;
```

gc	Specifies the gc to be deallocated.

References to sharable GCs are counted and a free request is generated to the server when the last user of a GC destroys it.

9.3. Merging Exposure Events into a Region

The X Toolkit provides the XtAddExposureToRegion utility function that merges Expose and GraphicsExpose events into a region that clients can process at once, rather than processing individual rectangles. (For further information about regions, see *Xlib – C Language X Interface.*)

To merge Expose and GraphicsExpose events into a region, use XtAddExposureToRegion.

```
void XtAddExposureToRegion(event, region)
    XEvent *event;
    Region region;
```

event	Specifies a pointer to the Expose or GraphicsExpose event.
region	Specifies the region object (as defined in X11/Xutil.h).

XtAddExposureToRegion computes the union of the rectangle defined by the exposure event and the specified region. Then, it stores the results back in region. If the event argument is not an Expose or GraphicsExpose event, XtAddExposureToRegion returns without an error and without modifying region.

This function is used by the exposure compression mechanism (see "Exposure Compression").

9.4. Translating Strings to Widget Instances

To translate a widget name to widget instance, use XtNameToWidget.

```
Widget XtNameToWidget(reference, names);
    Widget reference;
    String names;
```

reference	Specifies the widget to start searching from.
names	Specifies the fully qualified name of the desired widget.

The names argument contains the name of a widget with respect to the reference widget parameter. The names argument can contain more than one widget name for widgets that are not direct children of the reference widget. A dot (".") separates each component name.

XtNameToWidget looks for a widget whose name is the first component in the names parameter and who is a child (pop-up or normal) of the reference widget. It then uses that widget as the new reference and repeats the search after deleting the first component from the specified names argument. XtNameToWidget returns NULL if it cannot find the specified widget.

If more than one child of the reference widget matches the name, XtNameToWidget may return any of the children. The X Toolkit does not require that all children of a widget have unique names. If the names argument contains more than one component and if more than one child matches the first component, XtNameToWidget may return NULL if the single branch that it follows does not contained the named widget. That is, XtNameToWidget does not back up and follow other matching branches of the widget tree.

9.5. Translating Widget Coordinates

To translate an x-y coordinate pair from widget coordinates to root coordinates, use XtTranslateCoords.

```
void XtTranslateCoords(w, x, y, rootx_return, rooty_return)
    Widget w;
    Position x, y;
    Position *rootx_return, *rooty_return;
```

w	Specifies the widget.
x	
y	Specify the widget-relative coordinates.
rootx_return	
rooty_return	Returns the root-relative x and y coordinates.

While XtTranslateCoords is similar to XTranslateCoordinates, it does not generate a server request because all the required information already is in the widget's data structures.

9.6. Translating a Window to a Widget

To translate a window and display pointer into a widget instance, use XtWindowToWidget.

```
Widget XtWindowToWidget(display, window)
    Display *display;
    Window window;
```

display	Specifies the display on which the window is defined.
window	Specify the window for which you want the widget.

9.7. Handling Errors

The X Toolkit allows a client to register a procedure to be called whenever a fatal or non-fatal error occurs. This facility is intended for error reporting and logging but not for error correction or recovery.

Error and warning handlers are of type XtErrorHandler:

```
typedef void (*XtErrorHandler)();

void ErrorHandler(message)
   String message;
```

To register a procedure to be called on fatal error conditions, use XtSetErrorHandler.

```
void XtSetErrorHandler(handler)
    XtErrorHandler handler;
```

handler Specifies the new fatal error procedure. Fatal error handlers should not return.

The default error handler provided by the X Toolkit is _XtError. On UNIX-based systems, it prints the message to standard error and terminates the application.

To call the installed fatal error procedure, use XtError.

```
void XtError(message)
    String message;
```

message Specifies the error message to report.

To register a procedure to be called on non-fatal error conditions, use XtSetWarningHandler.

```
void XtSetWarningHandler(handler)
    XtErrorHandler handler;
```

handler Specifies the new non-fatal error procedure. Warning handlers usually return.

The default warning handler provided by the X Toolkit is _XtWarning. On UNIX-based systems, it prints the message to standard error and returns to the caller.

To call the installed non-fatal error procedure, use XtWarning.

```
void XtWarning(message)
    String message;
```

message Specifies the non-fatal error message to report.

Chapter 10

Event Handling

While X allows the reading and processing of events anywhere in an application, widgets in the X Toolkit neither directly read events nor grab the server or pointer. Widgets merely register procedures that are to be called when an event or class of events occurs in that widget.

A typical application consists of startup code followed by an event loop (see XtMainLoop, which reads events and dispatches them by calling the procedures that widgets have registered.

The event manager is a collection of routines to:

- Add or remove event sources other that X server events (in particular, timer interrupts and file input).
- Query the status of event sources.
- Add or remove procedures to be called when an event occurs for a particular widget.
- Enable and disable the dispatching of user-initiated events (keyboard and pointer events) for a particular widget.
- Constrain the dispatching of events to a cascade of ''pop-up'' widgets.
- Focus keyboard events within a composite widget to a particular child.
- Call the appropriate set of procedures currently registered when an event is read.

Most widgets do not need to call any of the event manager routines explicitly. The normal interface to X events is through the higher-level Translation Manager, which maps sequences of X events (with modifiers) into procedure calls. Applications rarely use any of the event manager routines besides XtMainLoop.

10.1. Adding and Deleting Additional Event Sources

While most applications are driven only by X events, some need to incorporate other sources of input into the X Toolkit event handling philosophy. The event manager provides routines to integrate notification of timer events and file data pending into this mechanism.

The next two functions provide input gathering from files. The application registers the files with the X Toolkit read routine. When input is pending on one of the files, the registered callback procedures are invoked.

10.1.1. Adding and Removing Input Sources

To register a new file for input, use XtAddInput.

```
XtInputId XtAddInput(source, condition, proc, client_data)
    int source;
    caddr_t condition;
    XtInputCallbackProc proc;
    caddr_t client_data;
```

source	Specifies the source file descriptor on a UNIX-based system or other operating system dependent device specification.
condition	Specifies the mask that indicates either a read, write, or exception condition or some operating system dependent condition.
proc	Specifies the procedure that is called when input is available.

client_data	Specifies the parameter to be passed to proc when input is available.

The XtAddInput function registers with the X Toolkit read routine a new source of events, which is usually file input but can also be file output. (The word "file" should be loosely interpreted to mean any sink or source of data.) XtAddInput also specifies the conditions under which the source can generate events. When input is pending on this source, the callback procedure is called.

Callback procedures that are called when there are file events are of type XtInputCallbackProc:

```
typedef void (*XtInputCallbackProc)();

void InputCallbackProc(client_data, source, id)
      caddr_t client_data;
      int source;
      XtInputId id;
```

client_data	Specifies the client data that was registered for this procedure in XtAddInput.
source	Specifies the source file descriptor generating the event.
id	Specifies the ID returned from the corresponding XtAddInput call.

To discontinue a source of input, use XtRemoveInput.

```
void XtRemoveInput(id)
    XtInputId *id;
```

id	Specifies the ID returned from the corresponding XtAddInput call.

The XtRemoveInput function causes the X Toolkit read routine to stop watching for input from the input source.

10.1.2. Adding and Removing Timeouts

The timeout facility notifies the application or the widget through a callback procedure that a specified time interval has elapsed. Timeout values are uniquely identified by an interval ID.

To create a timeout value, use XtAddTimeOut.

```
XtIntervalId XtAddTimeOut(interval, proc, client_data)
    unsigned long interval;
    XtTimerCallbackProc proc;
    caddr_t client_data;
```

interval	Specifies the time interval in milliseconds.
proc	Specifies the procedure to be called when time expires.
client_data	Specifies the parameter to be passed to proc when it is called.

The XtAddTimeOut function creates a timeout and returns an identifier for it. The timeout value is set to interval. The callback procedure will be called when the time interval elapses, after which the timeout is removed.

Callback procedures that are called when timeouts expire are of type XtTimerCallbackProc:

```
typedef void (*XtTimerCallbackProc)();

void TimerCallbackProc(client_data, id)
      caddr_t client_data;
      XtIntervalId *id;
```

client_data	Specifies the client data that was registered for this procedure in XtAddTimeOut.
id	Specifies the ID returned from the corresponding XtAddTimeOut call.

To clear a timeout value, use XtRemoveTimeOut.

```
void XtRemoveTimeOut(timer)
    XtIntervalId timer;
```

timer	Specifies the unique identifier for the timeout request to be destroyed.

XtRemoveTimeOut removes the timeout. Note that timeouts are automatically removed once they trigger.

10.2. Filtering X Events

The event manager provides two filters that can be applied to X user events. These filters screen out events that are redundant or that are temporarily unwanted.

10.2.1. Pointer Motion Compression

Widgets can have a hard time keeping up with pointer motion events. Further, they usually do not actually care about every motion event. To throw out redundant motion events, the widget class field compress_motion should be TRUE. When a request for an event would return a motion event, the Intrinsics check if there are any other motion events immediately following the current one, and, if so, skip all but the last of them.

10.2.2. Enter/Leave Compression

To throw out pairs of enter and leave events that have no intervening events, the widget class field compress_enter/leave should be TRUE. These enter and leave events will never be delivered to the client.

10.2.3. Exposure Compression

Many widgets will prefer to process a series of exposure events as a single expose region rather than as individual rectangles. Widgets with complex displays might use the expose region as a clip list in a graphics context, while widgets with simple displays might ignore the region entirely and redisplay their whole window or might get the bounding box from the region and redisplay only that rectangle.

In either case, these widgets do not care about getting partial expose events. If the compress_exposure field in the widget class structure is TRUE, the Event Manager calls the widget's expose procedure only once for each series of exposure events. In this case, all Expose events are accumulated into a region. When the Expose event with count zero is received, the Event Manager replaces the rectangle in the event with the bounding box for the region and calls the widget's expose procedure passing the (modified) exposure event and the region.

If compress_exposure is FALSE, the Event Manager will call the widget's expose procedure for every exposure event, passing the event and a region argument of NULL.

10.2.4. Setting and Checking the Sensitivity State of a Widget

To set the sensitivity state of a widget, use XtSetSensitive.

```
void XtSetSensitive(w, sensitive)
    Widget w;
    Boolean sensitive;
```

w	Specifies the widget.
sensitive	Specifies whether or not the widget should receive keyboard and pointer events.

Many widgets, especially those with callback lists that get executed in response to some user-initiated action (for example, clicking down or up), have a mode in which they take on a different appearance (for example, greyed out or stippled) and do not respond to user events.

This dormant state means the widget is "insensitive". If a widget is insensitive, the Event Manager does not dispatch any events to the widget with an event type of KeyPress, KeyRelease, ButtonPress, ButtonRelease, MotionNotify, EnterNotify, LeaveNotify, FocusIn, or FocusOut.

A widget can be insensitive because its sensitive field is FALSE or because one of its parents is insensitive, and, thus, the widget's ancestor_sensitive field also is FALSE. A widget may but does not need to distinguish these two cases visually.

XtSetSensitive first calls XtSetValues on the current widget with an argument list specifying that the sensitive field should change to the new value. It then recursively propagates the new value down the managed children tree by calling XtSetValues on each child to set the ancestor_sensitive to the new value if the new values for sensitive and the child's ancestor_sensitive are not the same. XtSetSensitive thus maintains the invariant:

- If parent has either sensitive or ancestor_sensitive FALSE, then all children have ancestor_sensitive FALSE.

XtSetSensitive calls XtSetValues to change sensitive and ancestor_sensitive. Therefore, when one of these changes, the widget's set_values procedure should take whatever display actions are needed, such as greying out or stippling the widget.

To check the current sensitivity state of a widget, use XtIsSensitive.

```
Boolean XtIsSensitive(w)
    Widget w;
```

w	Specifies the widget that is to be checked.

To indicate whether or not user input events are being dispatched, XtIsSensitive returns TRUE or FALSE. If both core.sensitive and core.ancestor_sensitive are TRUE, XtIsSensitive returns TRUE. Otherwise, it returns FALSE.

10.3. Adding and Removing X Event Handlers

Event handlers are procedures that are called when specified events occurs in a widget. Most widgets will not need to use event handlers explicitly. Instead, they use the Intrinsic's translation manager. Event handlers are of the type XtEventHandler:

```
typedef void (*XtEventHandler)();

void EventHandler(w, client_data, event)
    Widget w;
    caddr_t client_data;
    XEvent *event;
```

w	Specifies the widget that this event handler was registered with.
client_data	Specifies the client specific information registered with the event handler. This is usually NULL if the event handler is registered by the widget itself.
event	Specifies the triggering event.

To register an event handler procedure with the dispatch mechanism, use XtAddEventHandler.

```
void XtAddEventHandler(w, event_mask, nonmaskable, proc, client_data)
    Widget w;
    XtEventMask event_mask;
    Boolean nonmaskable;
    XtEventHandler proc;
    caddr_t client_data;
```

w	Specifies the widget for which this event handler is being registered.
event_mask	Specifies the event mask for which to call this procedure.
nonmaskable	Specifies whether this procedure should be called on the nonmaskable events. These are event of type GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage, and MappingNotify.
proc	Specifies the client event handler procedure.
client_data	Specifies additional data to be passed to the client's event handler.

The XtAddEventHandler function registers a procedure with the dispatch mechanism that is to be called when an event that matches the mask occurs on the specified widget. If the procedure is already registered, the specified mask is ORed into the existing mask. If the widget is realized, XtAddEventHandler calls XSelectInput, if necessary.

To remove a previously registered event handler, use XtRemoveEventHandler.

```
void XtRemoveEventHandler(w, event_mask, nonmaskable, proc, client_data)
    Widget w;
    XtEventMask event_mask;
    Boolean nonmaskable;
    XtEventHandler proc;
    caddr_t client_data;
```

w	Specifies the widget for which this procedure is registered.
event_mask	Specifies the event mask for which to unregister this procedure.
nonmaskable	Specifies the events for which to unregister this procedure.
proc	Specifies the event handler procedure registered.
client_data	Specifies the client data registered.

XtRemoveEventHandler stops the specified procedure from receiving the specified events. If the widget is realized, XtRemoveEventHandler calls XSelectInput, if necessary.

To stop a procedure from receiving any events (which will remove it from the widget's event_table entirely), call XtRemoveEventHandler with an event_mask of XtAllEvents and with nonmaskable TRUE.

10.3.1. Adding and Removing Event Handlers without Selecting Events

On occasions, clients need to register an event handler procedure with the dispatch mechanism without causing the server to select for that event. To do this, use XtAddRawEventHandler.

```
void XtAddRawEventHandler(w, event_mask, nonmaskable, proc, client_data)
    Widget w;
    XtEventMask event_mask;
    Boolean nonmaskable;
    XtEventHandler proc;
    caddr_t client_data;
```

w	Specifies the widget for which this event handler is being registered.
event_mask	Specifies the event mask for which to call this procedure.
nonmaskable	Specifies whether this procedure should be called on the nonmaskable events.
proc	Specifies the client event handler procedure.
client_data	Specifies additional data to be passed to the client's event handler.

This function has the same behavior as XtAddEventHandler, except that it does not affect the widget's mask and it never causes an XSelectInput for its events. Note that the widget might already have those mask bits set because of other non-raw, event handlers registered on it.

To remove a previously registered raw event handler, use XtRemoveRawEventHandler.

```
void XtRemoveRawEventHandler(w, event_mask, nonmaskable, proc, client_data)
    Widget w;
    XtEventMask event_mask;
    Boolean nonmaskable;
    XtEventHandler proc;
    caddr_t client_data;
```

w	Specifies the widget for which this procedure is registered.
event_mask	Specifies the event mask for which to unregister this procedure.
nonmaskable	Specifies the events for which to unregister this procedure.
proc	Specifies the event handler procedure registered.
client_data	Specifies the client data registered.

XtRemoveRawEventHandler stops the specified procedure from receiving the specified events. Because the procedure is a raw event handler, this will not affect the widget's mask and will never cause a call on XSelectInput.

10.4. Constraining Events to a Cascade of Widgets

Some widgets lock out any user input to the application except input to that widget. These are called ''modal'' widgets.

When a modal menu or modal dialog box is popped up using XtPopup, user events (that is, keyboard and pointer events) that occur outside the modal widget should be delivered to the modal widget or ignored. In no case should user events be delivered to a widget outside of the modal widget.

Menus can pop-up submenus and dialog boxes can pop-up further dialog dialog boxes to create a pop-up ''cascade''. In this case, user events should be delivered to one of several modal widgets in the cascade.

Display-related events should be delivered outside the modal cascade so that expose events and the like keep the application's display up-to-date. Any event that occurs within the cascade is delivered normally. Events that are delivered to the most recent spring-loaded shell in the cascade if they occur outside the cascade are called ''remap'' events and consist of the following events: KeyPress, KeyRelease, ButtonPress, and ButtonRelease.

Events that are ignored if they occur outside the cascade are: MotionNotify, EnterNotify, and LeaveNotify. All other events are delivered normally.

XtPopup uses the procedures XtAddGrab and XtRemoveGrab to constrain user events to a modal cascade and subsequently to remove a grab when the modal widget goes away. There is usually no need to call them explicitly.

To redirect user input to a modal widget, use XtAddGrab.

```
void XtAddGrab(w, exclusive, spring_loaded)
     Widget w;
     Boolean exclusive;
     Boolean spring_loaded;
```

w	Specifies the widget to add to the modal cascade.
exclusive	Specifies if user events should be dispatched exclusively to this widget or also to previous widgets in the cascade.
spring_loaded	Specifies if this widget was popped up because the user pressed a pointer button.

XtAddGrab appends the widget (and associated parameters) to the modal cascade. XtAddGrab checks that exclusive is TRUE if spring_loaded is TRUE. If not, it generates an error.

When XtDispatchEvent tries to dispatch a user event when at least one modal widget is in the widget cascade, it first determines if the event should be delivered. It starts at the most recent cascade entry and follows the cascade up to and including the most recent cascade entry added with the exclusive parameter TRUE.

This subset of the modal cascade is the active subset. User events that occur outside the widgets in this subset are ignored or remapped. Modal menus with submenus generally add a submenu widget to the cascade with exclusive FALSE. Modal dialog boxes that wish to restrict user input to the most deeply nested dialog box add a subdialog widget to the cascade with exclusive TRUE.

User events that occur within the active subset are delivered to the appropriate widget, which is usually a child or further descendant of the modal widget.

Regardless of where on the screen they occur, remap events are always delivered to the most recent widget in the active subset of the cascade that has spring_loaded TRUE (if any such widget exists).

To remove the redirection of user input to a modal widget, use XtRemoveGrab.

```
void XtRemoveGrab(w)
     Widget w;
```

w	Specifies the widget to remove from the modal cascade.

XtRemoveGrab removes widgets from the modal cascade starting at the most recent widget up to and including the specified widget. It issues an error if w is not on the modal cascade.

10.5. Focusing Events on a Child

To redirect keyboard input to a child of a composite widget without calling XSetInputFocus, use XtSetKeyboardFocus.

```
        XtSetKeyboardFocus(w, descendant)
            Widget w, descendant;
```

w	Specifies the widget for which the keyboard focus is to be set.
descendant	Specifies the widget to which the keyboard event is to be sent or None.

If a future keyboard event (KeyPress or KeyRelease) occurs on the specified widget, XtSet-KeyboardFocus causes. XtDispatchEvent to remap and send the event to the specified descendant widget.

If widget A is an ancestor of widget W and if no modal cascade has been created, a keyboard event is defined as occurring within W if one of the following focus conditions are true:

- W has the X server input focus.
- The event occurs within A or a descendent of A, and W has the keyboard focus for A.
- The event occurs within A, and no descendant of A has the keyboard focus for A, and the pointer is within W.

If widget A is an ancestor of widget W, and if a modal cascade exists, a keyboard event is defined as occurring within W if A is in the active subset of the modal cascade and if one of the previous focus conditions are true.

When W acquires the X input focus or the keyboard focus for one of its ancestors, a FocusIn event is generated for descendant if FocusNotify events have been selected by the descendant. Similarly, when W looses the X input focus or the keyboard focus for one of its ancestors, a FocusOut event is generated for descendant if FocusNotify events have been selected by the descendant.

Widgets that want the input focus may call XSetInputFocus explicitly. To allow outside agents to cause a widget to get the input focus, every widget exports an accept_focus procedure. Widgets interested in knowing when they lose the input focus must use the Xlib focus notification mechanism explicitly, typically by specifying translations for FocusIn and FocusOut events. Widgets that never want the input focus should set their accept_focus procedure to NULL.

Composite widgets have two additional functions:

- move_focus_to_next
- move_focus_to_prev

These procedures (which may be NULL) move the focus to the next child widget that wants it and to the previous child widget that wants it, respectively. The definition of next and previous is left to each individual widget. In addition, composite widgets are free to implement other procedures to move the focus between their children. Both move_focus_to_next and move_focus_to_prev should be entered in the widget class action table, so that they are available to translation specifications.

10.6. Querying Event Sources

The event manager provides several routines to examine and read events (including file and timer events) that are in the queue.

The next three functions handle X Toolkit equivalents of the XPending, XPeekEvent, and XNextEvent Xlib calls.

To determine if there are any events on the input queue, use XtPending.

```
        Boolean XtPending()
```

The XtPending returns a nonzero value if there are events pending from the X server or other input sources. If there are no events pending, it flushes the output buffer and returns a zero value.

To return the value from the head of the input queue without removing input from the queue, use XtPeekEvent.

```
        Boolean XtPeekEvent(event_return)
            XEvent *event_return;
```

event_return Returns the event information to the specified event structure.

If there is an event in the queue, XtPeekEvent fills in the event and returns a non-zero value. If no X input is on the queue, XtPeekEvent flushes the output buffer and blocks until input is available, possibly calling some timeout callbacks in the process. If the input is an event, XtPeekEvent fills in the event and returns a non-zero value. Otherwise, the input is for an alternate input source, and XtPeekEvent returns zero.

To return the value from the head of the input queue, use XtNextEvent.

```
        void XtNextEvent(event_return)
            XEvent *event_return;
```

event_return Returns the event information to the specified event structure.

If no input is on the X input queue, XtNextEvent flushes the X output buffer and waits for an event while looking at the other input sources and timeout values and calling any callback procedures triggered by them.

10.7. Dispatching Events

The X Toolkit provides functions that dispatch events to widgets or other application code. Every client interested in events on a widget uses XtAddEventHandler to register which events it is interested in and a procedure (event handler) that is to be called when the event happens in that window.

When an event is received, it is passed to a dispatching procedure. This procedure calls the appropriate event handlers and passes them the widget, the event, and client-specific data registered with each procedure. If there are no handlers for that event registered, the event is ignored and the dispatcher simply returns.

The order in which the handlers are called is not defined.

To send events to registered functions and widgets, use XtDispatchEvent. Usually, this procedure is not called by client applications (see XtMainLoop).

```
        void XtDispatchEvent(event)
            XEvent *event;
```

event Specifies a pointer to the event structure that is to be dispatched to the appropriate event handler.

The XtDispatchEvent function sends those events to those event handler functions that have been previously registered with the dispatch routine. The most common use of XtDispatchEvent is to dispatch events acquired with the XtNextEvent or XtPeekEvent procedure. However, it also can be used to dispatch user-constructed events. XtDispatchEvent also is responsible for processing grabs and keyboard focus.

10.8. Processing Input

To process input, an application can call XtMainLoop.

```
void XtMainLoop()
```

XtMainLoop first reads the next incoming file, timer, or X event by calling XtNextEvent. Then, it dispatches this to the appropriate registered procedure by calling XtDispatchEvent. This is the main loop of X Toolkit applications, and, as such, it does not return. Applications are expected to exit in response to some user action.

There is nothing special about XtMainLoop. It is simply an infinite loop that calls XtNextEvent then XtDispatchEvent.

Applications can provide their own version of this loop, which tests some global termination flag or tests that the number of top-level widgets is larger than 0 before circling back to the call to XtNextEvent.

10.9. Widget Exposure and Visibility

Every primitive widget and some composite widgets display data on the screen by means of raw X calls. Widgets cannot simply write to the screen and forget what they have done. They must keep enough state to redisplay the window or parts of it if a portion is obscured and then re-exposed.

10.9.1. Redisplay of a Widget: the expose procedure

The expose procedure for a widget class is of type XtExposeProc:

```
typedef void (*XtExposeProc)();

void ExposeProc(w, event, region)
     Widget w;
     XEvent *event;
     Region region;
```

w	Specifies the widget instance requiring redisplay.
event	Specifies the exposure event giving the rectangle requiring redisplay.
region	Specifies the union of all rectangles in this exposure sequence.

Redisplay of a widget upon exposure is the responsibility of the expose procedure in the widget's class record. If a widget has no display semantics, it can specify NULL for the expose field. Many composite widgets serve only as containers for their children and have no expose procedure.

Note

If the expose proc is NULL, XtRealizeWidget fills in a default bit gravity of NorthWestGravity before it calls the widget's realize proc.

If the widget's compress_exposure field is FALSE (see "Exposure Compression"), region will always be NULL. If the widget's compress_exposure field is TRUE, event will contain the bounding box for region.

A small simple widget (for example, Label) can ignore the bounding box information in the event and just redisplay the entire window. A more complicated widget (for example, Text) can use the bounding box information to minimize the amount of calculation and redisplay it does. A very complex widget will use the region as a clip list in a GC and ignore the event information.

The expose procedure is responsible for exposure of all superclass data as well as its own, This is because, in general, this operation cannot be cleanly broken up.

However, it often possible to anticipate the display needs of several levels of subclassing. For example, rather than separate display procedures for the widgets Label, Command, and Toggle, you could write a single display routine in Label that uses ''display state'' fields like:

```
Boolean invert
Boolean highlight
Dimension highlight_width
```

Label would have invert and highlight always FALSE and highlight_width zero(0). Command would dynamically set highlight and highlight_width, but it would leave invert always FALSE. Finally, Toggle would dynamically set all three.

In this case, the expose procedures for Command and Toggle inherit their superclass's expose procedure. For further information, see ''Inheriting Superclass Operations''.

10.9.2. Widget Visibility

Some widgets may use substantial computing resources to display data. However, this effort is wasted if the widget is not actually visible on the screen. That is, the widget can be obscured by another application or iconified.

The visible field in the Core widget structure provides a hint to the widget that it need not display data. This field is guaranteed TRUE (by the time an Expose event is processed) if the widget is visible and is usually FALSE if the widget is not visible.

Widgets can use or ignore the visible hint as they wish. If they ignore it, they should have visible_interest in their widget class record set FALSE. In such cases, the visible field is initialized TRUE and never changes. If visible_interest is TRUE, the Event Manager asks for VisibilityNotify events for the widget and updates the visible field accordingly.

10.10. Geometry Management – Sizing and Positioning Widgets

A widget does not directly control its size and location, which is the responsibility of the parent of that widget. Although the position of children is usually left up to their parent, the widgets themselves often have the best idea of their optimal sizes and, possibly, preferred locations.

To resolve physical layout conflicts between sibling widgets and between a widget and its parent, the X Toolkit provides the Geometry Management mechanism. Almost all composite widgets have a geometry manager (geometry_manager field in the widget class record) that is responsible for the size, position, and stacking order of the widget's children. The only exception are fixed boxes, which create their children themselves and can ensure that their children will never make a geometry request.

Widgets that wish to change their size, position, border width, or stacking depth must not use X calls directly. Instead, they must ask their parent's geometry manager to make the desired changes. When a child makes a request of the parent's geometry manager, the geometry manager can do one of the following:

- Allow the request
- Disallow the request

- Suggest a compromise

Geometry requests are always made by the child itself. Clients that wish to change the geometry of a widget should call XtSetValues on the appropriate geometry fields. Parents that wish to change the geometry of a child can use XtMoveWidget or XtResizeWidget at any time.

When the geometry manager is asked to change the geometry of a child, the geometry manager may also rearrange and resize any or all of the other children that it controls. The geometry manager can move children around freely using XtMoveWidget. When it resizes a child (that is, changes width, height, or border_width) other than the one making the request, it should do so by calling XtResizeWidget.

Often, geometry managers find that they can satisfy a request only if they can reconfigure a widget that they are not in control of (in particular, when the composite widget wants to change its own size). In this case, the geometry manager makes a request to its parent's geometry manager. Geometry requests can cascade this way to arbitrary depth.

Because such cascaded arbitration of widget geometry can involve extended negotiation, windows are not actually allocated to widgets at application startup until all widgets are satisfied with their geometry. See "Realizing Widgets" and "Creating Widgets" for more details.

10.10.1. Making General Geometry Manager Requests

To make a general geometry manager request from a widget, use XtMakeGeometryRequest.

```
XtGeometryResult XtMakeGeometryRequest(w, request, reply_return)
    Widget w;
    XtWidgetGeometry *request;
    XtWidgetGeometry *reply_return;
```

w	Specifies the widget ID of the widget that is making the request.
request	Specifies the desired widget geometry (size, position, border width, and stacking order).
reply_return	Returns the allowed widget size. If a widget is not interested in handling XtGeometryAlmost, the reply parameter can be NULL.

The return codes from geometry managers are:

```
typedef enum _XtGeometryResult {
        XtGeometryYes,
        XtGeometryNo,
        XtGeometryAlmost,
        XtGeometryDone,
} XtGeometryResult;
```

The XtWidgetGeometry structure is quite similar but not identical to the corresponding Xlib structure:

```
typedef unsigned long XtGeometryMask;

typedef struct {
        XtGeometryMask request_mode;
        Position x, y;
        Dimension width, height;
        Dimension border_width;
        Widget sibling;
```

```
            int stack_mode;
        } XtWidgetGeometry;
```

The request_mode definitions are from <X11/X.h>:

```
#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)
```

XtMakeGeometryRequest, in exactly the same manner as the Xlib routine XConfigureWindow, uses the request_mode to determine which fields in the XtWidget-Geometry structure you want to specify.

The stack_mode definitions are from <X11/X.h>:

```
#define Above             0
#define Below             1
#define TopIf             2
#define BottomIf          3
#define Opposite          4
#define XtSMDontChange    5
```

For definition and behavior of Above, Below, TopIf, BottomIf, and Opposite, see *Xlib – C Language X Interface.* XtSMDontChange indicates that the widget wants its current stacking order preserved.

The XtMakeGeometryRequest function performs the following:

- If the parent is not a subclass of Composite, or the parent's geometry_manager is NULL, it issues an error.
- If the widget's being_destroyed field is TRUE, it returns XtGeometryNo.
- If the widget x, y, width, height and border_width fields are all equal to the requested values, it returns XtGeometryYes.
- If the widget is unmanaged or the widget's parent is not realized, it makes the changes and returns XtGeometryYes. Otherwise, XtMakeGeometryRequest calls the parent's geometry_manager procedure with the given parameters.
- If the parent's geometry manager returns XtGeometryYes and if the widget is realized, it reconfigures the widget's window, setting its size, location, and stacking order as appropriate, by calling XConfigureWindow.
- If the geometry manager returns XtGeometryDone, it means that it has approved the change and, furthermore, has already done it. it does no configuring and changes the return value into XtGeometryYes. XtMakeGeometryRequest never returns XtGeometryDone.
- Finally, XtMakeGeometryRequest returns the resulting value from the parent's geometry manager.

10.10.2. Making Resize Requests

To make a simple resize request from a widget, you can use XtMakeResizeRequest as an alternative to XtMakeGeometryRequest.

```
XtGeometryResult XtMakeResizeRequest(w, width, height, width_return, height_return)
    Widget w;
    Dimension width, height;
    Dimension *width_return, *height_return
```

w	Specifies the widget.
width	Specifies the desired widget width.
height	Specifies the desired widget height.
width_return	Returns the allowed widget width.
height_return	Returns the allowed widget height.

XtMakeResizeRequest is a simple interface to XtMakeGeometryRequest. It creates a XtWidgetGeometry structure and specifies that width and height should change. The geometry manager is free to modify any of the other window attributes (position or stacking order) in order to satisfy the resize request. If the return value is XtGeometryAlmost, replyWidth and replyHeight contain a ''compromise'' width and height. If these are acceptable the widget should immediately make an XtMakeResizeRequest requesting the compromise width and height.

If the widget is not interested in XtGeometryAlmost replies, it can pass NULL for replyWidth and replyHeight.

10.10.3. Management of Child Geometry: the geometry_manager procedure

The geometry_manager procedure for a composite widget class is of type XtGeometryHandler:

```
typedef XtGeometryResult (*XtGeometryHandler)();

XtGeometryResult GeometryHandler(w, request, geometry_return)
    Widget w;
    XtWidgetGeometry *request;
    XtWidgetGeometry *geometry_return;
```

A class can inherit its superclass's geometry manager during class initialization.

A zero (0) bit in the request's mask field means that the child widget does not care about the value of the corresponding field. Then, the geometry manager can change it as it wishes. A one (1) bit means that the child wants that geometry element changed to the value in the corresponding field.

If the geometry manager can satisfy all changes requested, it updates the widget's x, y, width, height, and border_width appropriately, and then returns XtGeometryYes. The value of the preferred_return argument is undefined. The widget's window is moved and resized automatically by XtMakeGeometryRequest.

Homogeneous composite widgets often find it convenient to treat the widget making the request the same as any other widget, possibly reconfiguring it as part of its layout process. If it does this, it should return XtGeometryDone to inform XtMakeGeometryRequest that it does not need to do the configure itself. Although XtMakeGeometryRequest resizes the widget's window, it does not call the widget class's resize procedure if the geometry manager returns XtGeometryYes. The requesting widget must perform whatever resizing calculations are needed explicitly.

If the geometry manager chooses to disallow the request, the widget cannot change its geometry. The value of the preferred_return parameter is undefined, and the geometry manager returns XtGeometryNo.

Sometimes the geometry manager cannot satisfy the request exactly, but it may be able to satisfy what it considers a similar request. That is, it could satisfy only a subset of the requests (for example, size but not position) or a lesser request (for example, it cannot make the child as big as the request but it can make the child bigger than its current size). In such cases, the geometry manager fills in preferred_return with the actual changes it is willing to make, including a appropriate mask, and returns XtGeometryAlmost. If a bit in reply.request_mode is zero (0), the geometry manager will not change the corresponding value if the preferred_return is used immediately in a new request. If a bit is one (1), the geometry manager will change that element to the corresponding value in preferred_return. More bits may be set in preferred_return than in the original request if the geometry manager intends to change other fields should the child accept the compromise.

When XtGeometryAlmost is returned, the widget must decide if the compromise suggested in preferred_return is acceptable. If so, the widget must not change its geometry directly. Rather, it must make another call to XtMakeGeometryRequest.

If the next geometry request from this child uses the preferred_return box filled in by an XtGeometryAlmost return and if there have been no intervening geometry requests on either its parent or any of its other children, the geometry manager must grant the request. That is, if the child asks again right away with the returned geometry, it will get an answer of XtGeometryYes.

To return an XtGeometryYes, the geometry manager will frequently rearrange the position of other managed children. To do this, it should call the procedure XtMoveWidget described below. However, a few geometry managers sometimes may change the size of other managed children. To do this, they should call the procedures XtResizeWidget or XtConfigureWidget.

Geometry managers must not assume that the request and preferred_return arguments point to independent storage. The caller is permitted to use the same field for both, and the geometry manager must allocate its own temporary storage, if necessary.

10.10.4. Moving and Resizing Widgets

To move a sibling widget of the child making the geometry request, use XtMoveWidget.

```
void XtMoveWidget(w, x, y)
    Widget w;
    Position x;
    Position y;
```

w	Specifies the widget.
x	
y	Specifies the new widget coordinates.

XtMoveWidget writes the new x and y values into the widget and, if the widget is realized, issues an XMoveWindow call on the widget's window.

To resize a sibling widget of the child making the geometry request, use XtResizeWidget.

```
void XtResizeWidget(w, width, height, border_width)
     Widget w;
     Dimension width;
     Dimension height;
     Dimension border_width;
```

w Specifies the widget.

width
height
border_width Specify the new widget size.

XtResizeWidget returns immediately if the new width, height, and border_width are the same as the old values. Otherwise, XtResizeWidget writes the new width, height, and border_width values into the widget and, if the widget is realized, issues an XConfigureWindow call on the widget's window.

If the new width or height are different from the old values, XtResizeWidget calls the widget's resize procedure to notify it of the size change.

A geometry manager must not call XtResizeWidget on the child that is making the request.

To move and resize the sibling widget of the child making the geometry request, use XtConfigureWidget.

```
void XtConfigureWidget(w, x, y, width, height, border_width)
     Widget w;
     Position x;
     Position y;
     Dimension width;
     Dimension height;
     Dimension border_width;
```

w Specifies the widget.

x
y Specify the new widget position.

width
height
border_width Specify the new widget size.

XtConfigureWidget writes the new x, y, width, height, and border_width values into the widget and, if the widget is realized, makes an XConfigureWindow call on the widget's window.

If the new width or height are different from the old values, XtConfigureWidget calls the widget's resize procedure to notify it of the size change. Otherwise, it simply returns.

10.10.5. Querying Preferred Geometry

To query a widget's preferred geometry, use XtQueryGeometry.

```
XtGeometryResult XtQueryGeometry(w, intended, preferred_return)
    Widget w;
    XtWidgetGeometry *intended, *preferred_return;
```

w Specifies the child widget.

intended Specifies any changes the parent plans to make to the child's geometry (can be NULL).

*preferred_return*Returns the child widget's preferred geometry.

The parent that wants to know a child's preferred geometry sets any changes that it intends to make to the child's geometry in the corresponding fields of the intended structure, sets the corresponding bits in intended.request_mode and calls XtQueryGeometry.

XtQueryGeometry clears all bits in the preferred_return->request_mode and checks the query_geometry field of the specified widget's class record. If query_geometry is not NULL, XtQueryGeometry calls the query_geometry proc passing as arguments the specified widget, intended, and preferred_return structures. If the intended argument is NULL, it is replaced with a pointer to an XtWidgetGeometry structure with request_mode=0 before calling query_geometry.

The query_geometry procedure is of type XtGeometryHandler.

```
XtGeometryResult QueryGeometry(w, request, preferred_return)
    Widget w;
    XtWidgetGeometry *request;
    XtWidgetGeometry *preferred_return;
```

The query_geometry procedure is expected to examine the bits set in intended->request_mode, evaluate the preferred geometry of the widget, and store the result in preferred_return (setting the bits in preferred_return->request_mode corresponding to those geometry field that it cares about). If the proposed geometry change is acceptable without modification, the query_geometry procedure should return XtGeometryYes. If at least one field in preferred_return is different from the corresponding field in intended or if a bit was set in preferred_return that was not set in intended, the query_geometry procedure should return XtGeometryAlmost. If the preferred geometry is identical to the current geometry, the query_geometry procedure should return XtGeometryNo.

After calling the query_geometry proc or if the query_geometry field is NULL, XtQueryGeometry examines all the unset bits in preferred_return->request_mode and sets the corresponding fields in preferred_return to the current values from the widget instance. If CWStackMode is not set, the stack_mode field is set to XtSMDontChange. XtQueryGeometry then returns the value returned by the query_geometry procedure or XtGeometryYes if the query_geometry field is NULL.

Therefore, the caller can interpret a return of XtGeometryYes as not needing to evaluate the contents of reply and, more importantly, not needing to modify it's layout plans. A return of XtGeometryAlmost means either that both the parent and the child expressed interest in at least one common field, and the child's preference does not match the parent's intentions, or that the child expressed interest in a field that the parent might need to consider. A return value of XtGeometryNo means that both the parent and the child expressed interest in a field and that the child suggests that the field's current value is it's preferred value.

In addition, whether or not the caller ignores the return value or the reply mask, it is guaranteed that the reply structure contains complete geometry information for the child.

Parents are expected to call XtQueryGeometry in their layout routine and wherever else they may care after change_managed has been called. The changed_managed procedure may assume that the child's current geometry is it's preferred geometry. Thus, the child is still responsible for storing values into its own geometry during it's initialize proc.

10.10.6. Management of Size Changes: the resize procedure

A child can be involuntarily resized by its parent at any time. Widgets usually want to know when they have changed size so that they can re-layout their displayed data to match the new size. When a parent resizes a child, it calls XtResizeWidget. This function updates the geometry fields in the widget, configures the window if the widget is realized, and calls the child's resize procedure to notify the child. The resize procedure is of type XtWidgetProc:

```
void Resize(w)
    Widget w;
```

w Specifies the widget.

If a class need not recalculate anything when a widget is resized, it can specify NULL for the resize field in its class record. This is an unusual case and should only occur for widgets with very trivial display semantics.

The resize procedure takes a widget as its only argument. The x, y, width, height and border_width fields of the widget contain the new values.

The resize procedure should recalculate the layout of internal data as needed. (For example, a centered Label in a window that changes size should recalculate the starting position of the text.) The widget must obey resize as a command and must not treat it as a request. A widget must not issue an XtMakeGeometryRequest or XtMakeResizeRequest call from its resize procedure.

10.11. Selections

Arbitrary widgets (possibly not all in the same application) communicate with each other by means of the selection mechanism defined by the server and Xlib. For further information, see *Xlib – C Language X Interface*.

Chapter 11

Resource Management

Writers of widgets need to obtain a large set of resources at widget creation time. Some of the resources come from the resource database, some from the argument list supplied in the call to XtCreateWidget, and some from the internal defaults specified for the widget. Resources are obtained first from the argument list, then from the resource database for all resources not specified in the argument list, and lastly from the internal default, if needed.

A resource is a field in the widget record with a corresponding resource entry in the widget's resource list (or in a superclass's resource list). This means that the field is settable by XtCreateWidget (by naming the field in the argument list), by an entry in the default resource files (by using either the name or class), and by XtSetValues. In addition, it is readable by XtGetValues.

Not all fields in a widget record are resources. Some are for ''bookkeeping'' use by the generic routines (like managed and being_destroyed). Other can be for local bookkeeping, while still others are derived from resources (many GCs and Pixmaps).

11.1. Resource Lists

A resource entry specifies a field in the widget, the textual name and class of the field that argument lists and external resource files use to refer to the field, as well as a default value that the field should get if no value is specified. The declaration for the XtResource structure is:

```
typedef struct {
        String resource_name;
        String resource_class;
        String resource_type;
        Cardinal resource_size;
        Cardinal resource_offset;
        String default_type;
        caddr_t default_address;
} XtResource, *XtResourceList;
```

The resource_name field contains the name used by clients to access the field in the widget. By convention, it starts with a lower-case letter and is spelled almost identically to the field name, except (underbar, character) is replaced by (capital character). For example, the resource name for background_pixel is ''backgroundPixel''. Widget header files typically contain a symbolic name for each resource name. All resource names, classes, and types used by the Intrinsics are named in the file <X11/StringDefs.h>. The Intrinsic's symbolic resource names begin with XtN and are followed by the string name (for example, XtNbackgroundPixel for ''backgroundPixel'').

A resource class offers two functions:

- It isolates you from different representations that widgets can use for a similar resource.
- It lets you specify values for several actual resources with a single name. A resource class should be chosen to span a group of closely-related fields.

For example, a widget can have several pixel resources: background, foreground, border, block cursor, pointer cursor, and so on. Typically, the background defaults to ''white'' and everything else to ''black''. The resource class for each of these resources in the resource list should be chosen so that it takes the minimal number of entries in the resource database to make background "offwhite" and everything else "darkblue".

In this case, the background pixel should have a resource class of Background and all the other pixel entries a resource class of Foreground. Then, the resource file needs just two lines to change all pixels to "offwhite" or "darkblue":

```
        Background:          offwhite
        Foreground:          darkblue
```

Similarly, a widget may have several resource fonts (such as normal and bold), but all fonts should have the class Font. Thus, to change all fonts, simply requires a single line in the default file:

```
        Font:  6x13
```

By convention, resource classes are always spelled starting with a capital letter. Their symbolic names are preceded with XtC (for example, XtCBackground).

The resource_type field is the physical representation type of the resource. By convention, it starts with an upper-case letter and is spelled identically to the type name of the field. The resource type is used when resources are fetched, to convert from the resource database format (usually String) or the default resource format (just about anything, but often String) to the desired physical representation (see "Resource Conversions"). The Intrinsics define the following resource types:

XtRBoolean
XtRLongBoolean
XtRCallback
XtRColor
XtRCursor
XtRDefaultColor
XtRDisplay
XtREditMode
XtRFile
XtRFloat
XtRFont
XtRFontStruct
XtRFunction
XtRGeometry
XtRInt
XtRJustify
XtROrientation
XtRPixel
XtRPixmap
XtRPointer
XtRString
XtRStringTable
XtRTranslationTable
XtRWidget
XtRWindow

The resource_size field is the size of the physical representation in bytes and normally should be specified as "sizeof(*type*)" so that the compiler fills in the value.

The resource_offset is the offset in bytes of the field within the widget. The XtOffset macro should be used to retrieve this value.

The default_type field is the representation type of the default resource value. If default_type is different from resource_type and the default_type is needed, the resource manager invokes a conversion procedure from default_type to resource_type. Whenever possible, the default type should be identical to the resource type in order to minimize widget creation time.

The default_address field is the address of the default resource value. The default is used only if a resource is not specified in the argument list or in the resource database.

The routines XtSetValues and XtGetValues also use the resource list to set and get widget state. For further information, see "Obtaining Widget State" and "Setting Widget State".

Here is an abbreviated version of the resource list in the Label widget:

```
/* Resources specific to Label */
static XtResource resources[] = {
        {XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),
          XtOffset(LabelWidget, label.foreground), XtRString, "Black"},
        {XtNfont, XtCFont, XtRFontStruct, sizeof(XFontStruct *),
          XtOffset(LabelWidget, label.font),XtRString, "Fixed"},
        {XtNlabel, XtCLabel, XtRString, sizeof(String),
          XtOffset(LabelWidget, label.label), XtRString, NULL},
                                                .
                                                .
                                                .
}
```

The complete resource name for a field of a widget instance is the concatenation of the application name (from argv[0]) or the –name command-line option (see XtInitialize), the instance names of all the widget's parents, the instance name of the widget itself, and the resource name of the specified field of the widget. Likewise, the full resource class of a field of a widget instance is the concatenation of the application class (from XtInitialize), the widget class names of all the widget's parents (not the superclasses), the widget class name of the widget itself, and the resource name of the specified field of the widget.

11.2. Determining the Byte Offset

To determine the byte offset of a field within a structure, use XtOffset.

```
Cardinal XtOffset(pointer_type, field_name)
    Type pointer_type;
    Field field_name;
```

pointer_type	Specifies a type that is declared as a pointer to the structure.
field_name	Specifies the name of the field for which to calculate the byte offset.

XtOffset is usually used to determine the offset of various resource fields from the beginning of a widget.

11.3. Determining the Number of Elements

To determine the number of elements in a fixed-size array, use XtNumber.

```
Cardinal XtNumber(array)
    ArrayVariable array;
```

array	Specifies a fixed-size array.

XtNumber is used to pass the number of elements in argument lists, resources lists, and other counted arrays.

11.4. Superclass to Subclass Chaining of Resource Lists

The procedure XtCreateWidget gets resources as a "superclass-to-subclass" operation. That is, the resources specified in Core's resource list are fetched, then those in the subclass, and so on down to the resources specified for this widget's class.

In general, if a widget resource field is declared in a superclass, that field is included in the superclass's resource list and need not be included in the subclass's resource list. For example, the Core class contains a resource entry for background_pixel. Consequently, the implementation of ''Label'' need not also have a resource entry for background_pixel. However, a subclass, just by specifying a resource entry for that field in its own resource list, can override the resource entry for any field declared in a superclass. This is most often done to override the defaults provided in the superclass with new ones.

11.5. Obtaining Subresources

A widget does not do anything to get its own resources. Instead, XtCreateWidget does this automatically before calling the class initialize procedure.

Some widgets have subparts that are not widgets but for which the widget would like to fetch resources. For example, the Text widget fetches resources for its source and sink. Such widgets call XtGetSubresources to accomplish this.

```
void XtGetSubresources(w, base, name, class, resources, num_resources, args, num_args)
    Widget w;
    caddr_t base;
    String name;
    String class;
    XtResourceList resources;
    Cardinal num_resources;
    ArgList args;
    Cardinal num_args;
```

w	Specifies the widget that wants resources for a subpart.
base	Specifies the base address of the subpart data structure where the resources should be written.
name	Specifies the name of the subpart.
class	Specifies the class of the subpart.
resources	Specifies the resource list for the subpart.
num_resources	Specifies the number of resources in the resource list.
args	Specifies the argument list to override resources obtained from the resource database.
num_args	Specifies the number of arguments in the argument list. If the specified args is NULL, num_args must be zero (0). However, if num_args is zero (0), args is not referenced.

XtGetSubresources constructs a name/class list from the application name/class, the name/classes of all its ancestors, and the widget itself. Then, it appends to this list the name/class pair passed in. The resources are fetched from the argument list, the resource database, or the default values in the resource list. Then, they are copied into the subpart record.

11.6. Obtaining Application Resources

To retrieve resources that are not specific to a widget but apply to the overall application, use XtGetApplicationResources.

```
void XtGetApplicationResources(w, base, resources, num_resources, args, num_args)
    Widget w;
    caddr_t base;
    XtResourceList resources;
    Cardinal num_resources;
    ArgList args;
    Cardinal num_args;
```

w	Is currently ignored and can be specified as NULL.
base	Specifies the base address of the subpart data structure where the resources should be written.
resources	Specifies the resource list for the subpart.
num_resources	Specifies the number of resources in the resource list.
args	Specifies the argument list to override resources obtained from the resource database.
num_args	Specifies the number of arguments in the argument list. If the specified args is NULL, num_args must be zero (0). However, if num_args is zero (0), args is not referenced.

XtGetApplicationResources first reconstructs the application name and class and then retrieves the resources from the argument list, the resource database, or the resource list default values. After adding base to each address, the resources are copied into the address given in the resource list.

11.7. Resource Conversions

The X Toolkit provides a mechanism for registering representation converters that are automatically invoked by the resource fetching routines. The X Toolkit additionally provides and registers several commonly used converters.

This resource conversion mechanism serves several purposes:

- It permit user and application resource files to contain ASCII representations of non-textual values.
- It allows textual or other representations of default resource values that are dependent upon the display, screen, or color map, and thus must be computed at run-time.
- It caches all conversion source and result data. Conversions that require much computation or space (for example, string to translation table), or that require round trips to the server (for example, string to font or color) are performed only once.

11.7.1. Predefined Resource Converters

The X Toolkit defines all the representations used in the Core, Composite, Constraint, and Shell widgets. Furthermore, it registers resource converters from string to all these representations.

The X Toolkit registers converters for XtRString to the following representations:

```
XtRBoolean                      XtRFont
XtRLongBoolean                  XtRFontStruct
XtRCursor                       XtInt
XtRDisplay                      XtPixel
XtRFile
```

11.7.2. Writing a New Resource Converter

Type converters use pointers to XrmValue structures (defined in X11/Xresource.h) for input and output values.

```
typedef struct {
        unsigned int size;
        caddr_t addr;
} XrmValue, *XrmValuePtr;
```

A resource converter is a procedure of type XtConverter:

```
typedef void (*XtConverter)();

void Converter(args, num_args, from, to)
     XrmValue *args;
     Cardinal *num_args;
     XrmValue *from;
     XrmValue *to;
```

args	Specifies a list of additional XrmValue arguments to the converter if additional context is needed to perform the conversion. For example, the string to font converter needs the widget's screen, or the string to pixel converter needs the widget's screen and color map. This argument is often NULL.
num_args	Specifies the number of additional XrmValue arguments. This argument is often 0.
from	Specifies the value to convert.
to	Specifies the descriptor to use to return the converted value.

Type converters should perform the following actions:

- Check to see that the number of arguments passed is correct.
- Attempt the type conversion.
- If successful, return a pointer to the data in the to parameter. Otherwise, optionally call XtWarning and return.

Most type converters just take the data described by the specified from argument and return data by writing into the specified to argument. A few need other information, which is available in the specified args.

A type converter can invoke another type converter. This allows differing sources which may convert into a common intermediate result to make maximum use of the type converter cache.

Note that the address written to to.addr cannot be a local variable of the converter because this will disappear when the converter returns. It should be a pointer to a static variable, as in the following example where screenColor is returned.

The following is an example of a converter that takes a string and converts it to a Pixel:

```
static void CvtStringToPixel(args, num_args, fromVal, toVal)
        XrmValuePtr args;
        Cardinal *num_args;
        XrmValuePtr fromVal;
        XrmValuePtr toVal;
{
        static XColor screenColor;
```

```
                XColor exactColor;
                Screen *screen;
                Colormap colormap;
                Status status;
                char message[1000];

                if (*num_args != 2)
                XtError("String to pixel conversion needs screen and colormap arguments");

                screen = *((Screen **) args[0].addr);
                colormap = *((Colormap *) args[1].addr);

                status = XAllocNamedColor(DisplayOfScreen(screen), colormap,
                  (String) fromVal->addr, &screenColor, &exactColor);
                if (status == 0) {
                  sprintf(message, "Cannot allocate colormap entry for %s",
                    (String) fromVal->addr);
                XtWarning(message);
                } else {
                  (*toVal).addr = &(screenColor.pixel);
                  (*toVal).size = sizeof(Pixel);
                }
        };
```

11.7.3. Registering a New Resource Converter

To register a new converter, use the procedure XtAddConverter.

```
void XtAddConverter(from_type, to_type, converter, convert_args, num_args)
   String from_type;
   String to_type;
   XtConverter converter;
   XtConvertArgList convert_args;
   Cardinal num_args;
```

from_type	Specifies the source type.
to_type	Specifies the destination type.
converter	Specifies the type converter procedure.
convert_args	Specifies how to compute the additional arguments to the converter. Most type converters have none, so convert_args is NULL.
num_args	Specifies the number of additional arguments to the converter. Most type converters have none, so num_args is 0.

For the few type converters that need additional arguments, the X Toolkit conversion mechanism provides a method of specifying how these arguments should be computed. The enumerated type XtAddressMode and the structure XtConvertArgRec specify how each argument is derived. These are defined in the <X11/Convert.h> header file.

```
typedef enum {
        /* address mode              parameter representation */
          XtAddress,                 /* address */
          XtBaseOffset,              /* offset */
          XtImmediate,               /* constant */
        XtResourceString,         /* resource name string */
        XtResourceQuark           /* resource name quark */
} XtAddressMode;

typedef struct {
        XtAddressMode address_mode;
        caddr_t address_id;
        Cardinal size;
} XtConvertArgRec, *XtConvertArgList;
```

The address_mode field specifies how the address_id field should be interpreted. XtAddress causes address_id to be interpreted as the address of the data. XtBaseOffset causes address_id to be interpreted as the offset from the widget base. XtImmediate causes address_id to be interpreted as a 4-byte constant. XtResourceString causes address_id to be interpreted as the name of a resource that is to be converted into an offset from widget base. XtResourceQuark is an internal compiled form of an XtResourceString.

The size field specifies the length of the data in bytes.

Here is the code used to register the CvtStringToPixel routine shown above:

```
static XtConvertArgRec colorConvertArgs[] = {
        {XtBaseOffset, (caddr_t) XtOffset(Widget, core.screen),  sizeof(Screen *)},
        {XtBaseOffset, (caddr_t) XtOffset(Widget, core.colormap),sizeof(Colormap)}
};

XtAddConverter(XtRString, XtRPixel, CvtStringToPixel,
   colorConvertArgs, XtNumber(colorConvertArgs));
```

The conversion argument descriptors colorConvertArgs and screenConvertArg are predefined for you. The screenConvertArg descriptor puts the widget's screen field into args[0]. The colorConvertArgs descriptor puts the widget's screen field into args[0], and the widget's colormap field into args[1].

It might seem easier to just create a descriptor that puts the widget's base address into args[0], and do your own indexing off that in the conversion routine. But you should not. If you constrain the dependencies of your conversion procedure to the minimum possible, you improve the chance that subsequent conversions will find what they need in the conversion cache. Then, you decrease the size of the cache by having fewer but more widely applicable entries.

11.7.4. Invoking Resource Converters

All resource-fetching routines (for example, XtGetSubresources, XtGetApplicationResources, and so on) call resource converters if the user specifies a resource that is a different representation from the desired representation, or if the widget's default resource value representation is different from the desired representation.

To invoke resource conversions, use XtConvert or XtDirectConvert.

The definition for XtConvert is:

```
void XtConvert(w, from_type, from, to_type, to_return)
    Widget w;
    String from_type;
    XrmValuePtr from;
    String to_type;
    XrmValuePtr to_return;
```

w	Specifies the widget to use for additional arguments (if any are needed).
from_type	Specifies the name of the source type.
from	Specifies the value to be converted.
to_type	Specifies the name of the destination type.
to_return	Returns the converted value.

XtConvert looks up the type converter registered to convert from_type to to_type and computes any additional arguments needed. It then calls XtDirectConvert.

The definition for XtDirectConvert is:

```
void XtDirectConvert(converter, args, num_args, from, to_return)
    XtConverter converter;
    XrmValuePtr args;
    Cardinal num_args;
    XrmValuePtr from;
    XrmValuePtr to_return;
```

converter	Specifies the widget to use for additional arguments (if any are needed).
args	Specifies the additional arguments needed to perform the conversion (often NULL).
num_args	Specifies the number of additional arguments (often 0).
from	Specifies the value to be converted.
to_return	Returns the converted value.

XtDirectConvert looks in the converter cache to see if this conversion procedure has been called with the specified arguments. If so, it just returns a descriptor for information stored in the cache. Otherwise, it calls the converter and enters the result in the cache.

11.8. Reading and Writing Widget State

Any resource field in a widget can be read or written by a client. On a write, the widget decides what changes it will actually allow and updates all derived fields appropriately.

11.8.1. Obtaining Widget State

To retrieve the current value of a resource associated with a widget instance, use XtGetValues.

```
void XtGetValues(w, args, num_args)
    Widget w;
    ArgList args;
    Cardinal num_args;
```

w	Specifies the widget.

args	Specifies a variable length argument list of name/address pairs that contain the resource name and the address to store the resource value into. The argument names in args are dependent on the widget.
num_args	Specifies the number of arguments in argument list.

XtGetValues starts with the resources specified for the core widget fields and proceeds down the subclass chain to the widget.

The value field of a passed Arg should contain the address into which to store the corresponding resource value.

If the widget's parent is a subclass of constraintWidgetClass, XtGetValues then fetches the values for any constraint resources requested. It starts with the constraint resources specified for constraintWidgetClass and proceeds down to the subclass chain to the parent's constraint resources.

Finally, the get_values_hook procedures, if non-NULL, are called in superclass-to-subclass order after all the resource values have been fetched by XtGetValues. This permits a subclass to provide non-widget resource data by means of the GetValues mechanism.

11.8.1.1. Widget Subpart Resource Data: the get_values_hook procedure

Widgets that have subparts can return the resource values by using XtGetValues and supplying a get_values_hook procedure. The get_values_hook procedure is of type XtArgsProc:

```
void get_values_hook(w, args, num_args)
    Widget w;
    ArgList args;
    Cardinal *num_args;
```

w	Specifies the widget whose non-widget resource values are to be retrieved.
args	Specifies the argument list that was passed to XtCreateWidget.
num_args	Specifies the number of arguments in the argument list.

11.8.1.2. Obtaining Widget Subpart State

To retrieve the current value of a non-widget resource data associated with a widget instance, use XtGetSubvalues. For a discussion of non-widget subclass resources resources, see ''Obtaining Subresources''.

```
void XtGetSubvalues(base, resources, num_resources, args, num_args)
    caddr_t base;
    XtResourceList resources;
    Cardinal num_resources;
    ArgList args;
    Cardinal num_args;
```

base	Specifies the base address of the subpart data structure from which the resources should be retrieved.
resources	Specifies the non-widget resources list.
num_resources	Specifies the number of resources in the resource list.
args	Specifies a variable length argument list of name/address pairs that contain the resource name and the address to store the resource value into. The arguments and values passed in args are dependent on the subpart. The storage for argument values that are pointed to by args must be deallocated by the application

when no longer needed.

num_args Specifies the number of arguments in argument list.

11.8.2. Setting Widget State

To modify the current value of a resource associated with a widget instance, use XtSetValues.

```
void XtSetValues(w, args, num_args)
    Widget w;
    ArgList args;
    Cardinal num_args;
```

w Specifies the widget.

args Specifies a variable length argument list of name/value pairs that contain the resources to be modified and their new values. The resources and values passed are dependent on the widget being modified.

num_args Specifies the number of resources in the argument list.

XtSetValues starts with the resources specified for the core widget fields and proceeds down the subclass chain to the widget. At each stage, it writes the new value (if specified by one of the arguments) or the existing value (if no new value is specified) to a new widget data record.

XtSetValues then calls the set_values procedures for the widget in ''superclass-to-subclass'' order. If the widget has non-NULL set_values_hook fields, these are called with the arguments immediately after the corresponding set_values procedure. This procedure permits subclasses to set non-widget data using the SetValues mechanism.

If the widget's parent is a subclass of constraintWidgetClass, XtSetValues also updates the widget's constraints. It starts with the constraint resources specified for constraintWidgetClass and proceeds down the subclass chain to the parent's class. At each stage, it writes the new value or the existing value to a new constraint record. It then calls the constraint set_values procedures from constraintWidgetClass down to the parent's class. The constraint set_values procedures are called with widget arguments (as for all set_values procs, not just the constraint record arguments), so that they can make adjustments to the desired values based on full information about the widget.

XtSetValues determines if a geometry request is needed by comparing the current widget to the new widget. If any geometry changes are required, it makes the request, and the geometry manager returns XtGeometryYes, XtGeometryAlmost, or XtGeometryNo. If XtGeometryYes, XtSetValues calls the widget's resize procedure. If XtGeometryNo, XtSetValues resets the geometry fields to their original values. If XtGeometryAlmost, XtSetValues calls the set_values_almost procedure, which determines what should be done and writes new values for the geometry fields into the new widget. XtSetValues then repeats this process, deciding once more whether the geometry manager should be called.

Finally, if any of the set_values procedures returned TRUE, XtSetValues causes the widget's expose procedure to be invoked by calling XClearArea on the widget's window.

11.8.2.1. Widget State: the set_values procedure

The set_values procedure for a widget class is of type XtSetValuesFunc:

```
typedef Boolean (*XtSetValuesFunc)();

Boolean SetValuesFunc(current, request, new)
    Widget current;
    Widget request;
    Widget new;
```

current	Specifies the existing widget.
request	Specifies a copy of the widget asked for by the XtSetValues call before any class set_values procedures have been called.
new	Specifies a copy of the widget with the new values that are actually allowed.

The set_values procedure should recompute any field derived from resources that are changed (for example, many GCs depend upon foreground and background). If no recomputation is necessary and if none of the resources specific to a subclass require the window to be redisplayed when their values are changed, then you can specify NULL for the set_values field in the class record.

Like the initialize procedure, set_values mostly deals only with the fields defined in the subclass, but it has to resolve conflicts with its superclass, especially conflicts over width and height. In this case, though, the "reference" widget is "request", not "new".

"New" starts with the values of "request" but has been modified by any superclass set_values procedures. A widget need not refer to "request" unless it must resolve conflicts between "current" and "new". Any changes that the widget wishes to make should be made in "new". XtSetValues will copy the "new" values back into the "current" widget instance record after all class set_values procedures have been called.

Finally, the set_values procedure must return a Boolean that indicates whether the widget needs to be redisplayed. Note that a change in the geometry fields alone does not require the set_values procedure to return TRUE; the X server will eventually generate an Expose event, if necessary. After calling all the set_values procedures, XtSetValues will force a redisplay (by calling XClearArea) if any of the set_values procedures returned TRUE. Therefore, a set_values procedure should not try to do its own redisplaying.

It is permissible to call XtSetValues before a widget is realized. Therefore, the set_values proc must not assume that the widget is realized.

11.8.2.2. Widget State: the set_values_almost procedure

The set_values_almost procedure for a widget class is of type XtAlmostProc:

```
typedef void (*XtAlmostProc)();

void AlmostProc(w, new_widget, request, reply)
    Widget w;
    Widget new_widget;
    XtWidgetGeometry *request;
    XtWidgetGeometry *reply;
```

w	Specifies the widget on which the geometry change is requested.
new_widget	The return value, with relevant geometry fields modified based on the geometry requests.
request	Specifies the original geometry request that was sent to the geometry manager that returned XtGeometryAlmost.

reply	Specifies the compromise geometry that was returned by the geometry manager that returned XtGeometryAlmost.

Most classes inherit this operation from their superclass by copying the Core set_values_almost procedure in their class_initialize procedure. The Core's set_values_almost procedure simply accepts the compromise suggested.

The set_values_almost procedure is called when a client tries to set a widget's geometry by means of a call to XtSetValues, and the geometry manager cannot satisfy the request but instead returns XtGeometryAlmost and a compromise geometry. The set_values_almost procedure takes the original geometry and the compromise geometry and determines whether the compromise is acceptable or a different compromise might work. It returns its results in the new_widget parameter, which will then be sent back to the geometry manager for another try.

11.8.2.3. Widget State: the constraint set_values procedure

The constraint set_values procedure is of type XtSetValuesFunc. The values passed to the parent's constraint set_values procedure are the same as those passed to the child's class set_values procedure. A class can specify NULL for the set_values field of the ConstraintPart if it need not compute anything.

The constraint set_values procedure should recompute any constraint fields derived from constraint resource that are changed. Further, it should modify the widget fields as appropriate. For example, if a constraint for the maximum height of a widget is changed to a value smaller than the widget's current height, the constraint set_values procedure should reset the height field in the widget.

11.8.2.4. Setting Widget Subpart State

To set the current value of a non-widget resource associated with a widget instance, use XtSetSubvalues. For a discussion of non-widget subclass resources, see ''Obtaining Subresources''.

```
void XtSetSubvalues(base, resources, num_resources, args, num_args)
    caddr_t base;
    XtResourceList resources;
    Cardinal num_resources;
    ArgList args;
    Cardinal num_args;
```

base	Specifies the base address of the subpart data structure where the resources should be written.
resources	Specifies the current non-widget resources values.
num_resources	Specifies the number of resources in the resource list.
args	Specifies a variable length argument list of name/value pairs that contain the resources to be modified and their new values. The resources and values passed are dependent on the subpart of the widget being modified.
num_args	Specifies the number of resources in argument list.

11.8.2.5. Widget Subpart Resource Data: the set_values_hook procedure

Widgets that have a subpart can set the resource values by using XtSetValues and supplying a set_values_hook procedure. The set_values_hook procedure for a widget class is of type XtArgsFunc:

```
typedef Boolean (*XtArgsFunc)();

Boolean ArgsFunc(w, args, num_args)
    Widget w;
    ArgList args;
    Cardinal *num_args;
```

w	Specifies the widget whose non-widget resource values are to be changed.
args	Specifies the argument list that was passed to XtCreateWidget.
num_args	Specifies the number of arguments in the argument list.

Chapter 12

Translation Management – Handling User Input

Except under unusual circumstances, widgets do not hardwire the mapping of user events into widget behavior by using the Event Manager. Instead, they provide a user-overridable default mapping of events into behavior.

The translation manager provides an interface to specify and manage the mapping of X Event sequences into widget-supplied functionality. The simplest example would be to call procedure Abc when key "y" is pressed.

The translation manager uses two kinds of tables to perform translations. The "action table", which is in the widget class structure, specifies the mapping of externally available procedure name strings to the corresponding procedure implemented by the widget class. The "translation table", which is in the widget class structure, specifies the mapping of event sequence to procedure names.

The translation table in the class structure can be over-ridden for a specific widget instance by supplying a different translation table for the widget instance.

12.1. Action Tables

All widget class records contain an action table. In addition, an application can register its own action tables with the translation manager, so that the translation tables it provides to widget instances can access application functionality. The translation action_proc procedure is of type XtActionProc:

```
typedef void (*XtActionProc)();

void ActionProc(w, event, params, num_params)
    Widget w;
    XEvent *event;
    String *params;
    Cardinal *num_params;
```

w	Specifies the widget that caused the action to be called.
event	Specifies the event that caused the action to be called. If the action is called after a sequence of events, then the last event in the sequence is used.
params	Specifies a pointer to the list of strings that were specified in the translation table as arguments to the action.
num_params	Specifies the number of arguments specified in the translation table.

```
typedef struct _XtActionsRec {
        String action_name;
        XtActionProc action_proc;
} XtActionsRec, *XtActionList;
```

The action_name field is the name that you use in translation tables to access the procedure. The action_proc field is a pointer to a procedure that implements the functionality.

For example, the Command widget has procedures to:

- Set the command button to indicate it is activated
- Unset the button back to its normal mode
- Highlight the button borders
- Unhighlight the button borders
- Notify any callbacks that the button has been activated

The action table for the Command widget class makes these functions available to translation tables written for Command or any subclass. The string entry is the name used in translation tables. The procedure entry (usually spelled identically to the string) is the name of the C procedure that implements that function:

```
XtActionsRec actionTable[] = {
        {"Set",         Set},
        {"Unset",       Unset},
        {"Highlight",   Highlight},
        {"Unhighlight", Unhighlight}
        {"Notify",      Notify},
};
```

12.1.1. Registering Action Tables

To make functionality available by declaring an action table and registering this with the translation manager, use XtAddActions.

```
void XtAddActions(actions, num_actions)
    XtActionList actions;
    Cardinal num_actions;
```

actions	Specifies the action table to register.
num_args	Specifies the number of entries in actions.

The X Toolkit registers an action table for MenuPopup and MenuPopdown as part of X Toolkit initialization.

12.1.2. Translating Action Names to Procedures

The translation manager uses a simple algorithm to convert the name of procedure specified in a translation table into the actual procedure specified in an action table. It performs a search for the name in the following tables:

- The widget's class action table for the name
- The widget's superclass action table, and on up the superclass chain
- The action tables registered with XtAddActions, from the most recently added table to the oldest table.

As soon as it finds a name, it stops the search. If it cannot find a name, the translation manager generates an error.

12.2. Translation Tables

All widget instance records contain a translation table, which is a resource with no default value. A translation table specifies what action procedures are invoked for an event or a sequence of events. It is a string containing a list of translations from an event (or event sequence) into one or more procedure calls. The translations are separated from one another by new-line characters (ASCII LF).

For example, the default behavior of Command is:

- Highlight on enter window
- Unhighlight on exit window
- Invert on left button down
- Call callbacks and reinvert on left button up

Command's default translation table is:

```
static String defaultTranslations =
        "<EnterWindow>:Highlight()\n\
        <LeaveWindow>:Unhighlight()\n\
        <Btn1Down>: Set()\n\
        <Btn1Up>:     Notify() Unset()";
```

For details on the syntax of translation tables, see Appendix B.

The tm_table field of the CoreClass record should be filled in at static initialization time with the string containing the class's default translations. If a class wishes to just inherit its superclass's translations, it can store the special value XtInheritTranslations into tm_table. After the class initialization procedures have been called, the Intrinsics compile this translation table into an efficient internal form. Then, at widget creation time, this default translation table will be used for any widgets that have not had their core translations field set by the resource manager or the initialize procedures.

The resource conversion mechanism takes care of automatically compiling string translation tables that are resources. If a client uses translation tables that are not resources, it must compile them itself using XtParseTranslations.

The X Toolkit uses the compiled form of the translation table to register the necessary events with the event manager. Widgets need do nothing other than specify the action and translation tables for events to be processed by the translation manager.

12.3. Merging Translation Tables

Sometimes an application needs to destructively or non-destructively add its own translations to a widget's translation. For example, a window manager provides functions to move a window. It normally may moves the window when any pointer button is pressed down in a title bar. It allows the user to specify other translations for the middle or right button down in the title bar, but it ignores any user translations for left button down.

To accomplish this, the window manager first should create the title bar and then should merge the two translation tables into the title bar's translations. One translation table contains the translations that the window manager wants only if the user has not specified a translation for a particular event (or event sequence). The other translation table contains the translations that the window manager wants regardless of what the user has specified.

Three X Toolkit functions support this merging:

XtParseTranslationTable	Compiles a translation table.
XtAugmentTranslations	Merges (non-destructively) a compiled translation table into a widget's compiled translation table.
XtOverrideTranslations	Merges destructively a compiled translation table into a widget's compiled translation table.

To compile a translation table, use XtParseTranslationTable.

```
XtTranslations XtParseTranslationTable(table)
    String table;
```

table	Specifies the translation table to compile.

XtParseTranslationTable compiles the translation table into the opaque internal representation (of type XtTranslations).

Tô merge new translations into an existing translation table, use XtAugmentTranslations.

```
void XtAugmentTranslations(w, translations)
    Widget w;
    XtTranslations translations;
```

w	Specifies the widget to merge the new translations into.
translations	Specifies the compiled translation table to merge in.

XtAugmentTranslations non-destructively merges the new translations into the existing widget translations. If the new translations contain an event or event sequence that already exists in the widget's translations, the new translation is ignored.

To overwrite existing translations with new translations, use XtOverrideTranslations.

```
void XtOverrideTranslations(w, translations)
    Widget w;
    XtTranslations translations;
```

w	Specifies the widget to merge the new translations into.
translations	Specifies the compiled translation table to merge in.

XtOverrideTranslations destructively merges the new translations into the existing widget translations. If the new translations contain an event or event sequence that already exists in the widget's translations, the new translation is merged in and override the widget's translation.

Appendix A

Resource File Format

A resource file contains text representing the default resource values for an application or set of applications. The resource file is an ASCII text file that consists of a number of lines with the following EBNF syntax:

```
Xdefault        = {line "\n"}.
line            = (comment | production).
comment         = "!" string.
production      = resourcename ":" string.
resourcename    = ["*"] name {("." | "*") name}.
string          = {<any character not including eol>}.
name            = {"A"-"Z" | "a"-"z" | "0"-"9"}.
```

If the last character on a line is a backslash (\), that line is assumed to continue on the next line.

To include a new-line character in a string, use '\n'.

Appendix B

Translation Table File Syntax

A translation table file is an ASCII text file.

Notation

Syntax is specified in EBNF notation, where:

[a]

means either nothing or ''a''.

{ a }

means 0 or more occurrences of ''a''

All terminals are enclosed in "double" quotes. Informal descriptions are enclosed in <angle> brackets.

Syntax

The syntax of the translation table file is:

translationTable	= [production { "\n" production }]
production	= lhs ":" rhs
lhs	= (event | keyseq) { "," (event | keyseq) }
keyseq	= """ keychar {keychar} """
keychar	= ["^" | "$"] <ascii character>
event	= [modifier_list] "<"event_type">" ["(" count["+"] ")"] {detail}
modifier_list	= [!] modifier {modifier} | "None"
modifier	= [~] modifier_name
count	= (2 | 3 | 4 | ...)
modifier_name	= <see ModifierNames table below>
event_type	= <see Event Types table below>
detail	= <event specific details>
rhs	= { name "(" [params] ")" }
name	= namechar { namechar }
namechar	= { "a"-"z" | "A"-"Z" | "0"-"9" | "$" | "_" }
params	= string {"," string}.
string	= quoted_string | unquoted_string
quoted_string	= """ {<ascii character>} """
unquoted_string	= {<ascii character except space, tab, ",", newline, ")">}

Informally, the productions are an event specifier on the left (terminated with a colon) and a list of action specifications on the right (terminated with a newline).

The information on the left specifies the X Event, complete with modifiers and detail fields, while that on the right specifies what to do when that event is detected. An action is the name of an exported function. The parameters are strings.

It is often convenient to include newlines in a translation table to make it more readable. In C, the newline should be preceded by a backslash (\):

```
"<Btn1Down>: DoSomething()\n\
<Btn2Down>:  DoSomethingElse()"
```

Modifier Names

The Modifier field is used to specify normal X keyboard and button modifier mask bits. If the modifier_list has no entries and is not "None", it means "don't care" on all modifiers. If any modifiers are specified, and "!" is not specified, it means that the listed modifiers must be in the correct state and "don't care" about any other modifiers. If "!" is specified at the beginning of the modifier list, it means that the listed modifiers must be in the correct state and no other modifiers can be asserted. If a modifier is preceded by a "~" it means that that modifier must not be asserted. If "None" is specified, it means no modifiers can be asserted. Briefly:

No Modifiers:	None <event> detail
Any Modifiers:	<event> detail
Only these Modifiers:	! mod1 mod2 <event> detail
These modifiers and any others:	mod1 mod2 <event> detail

Modifier	Meaning
c	Control Key
Ctrl	Control Key
s	Shift Key
Shift	Shift Key
m	Modifier 1
Meta	Modifier 1
l	Lock Key
Lock	Lock Key
1	Modifier 1
Mod1	Modifier 1
2	Modifier 2
Mod2	Modifier 2
3	Modifier 3
Mod3	Modifier 3
4	Modifier 4
Mod4	Modifier 4
5	Modifier 5
Mod5	Modifier 5
ANY	Any combination

Event Types

The EventType field describes XEvent types. The following are the currently defined EventType values:

Type	Meaning
Key	KeyPress
KeyDown	KeyPress
KeyUp	KeyRelease
BtnDown	ButtonPress
BtnUp	ButtonRelease
Motion	MotionNotify
BtnMotion	MotionNotify with any button down
Btn1Motion	MotionNotify with button 1 down
Btn2Motion	MotionNotify with button 2 down
Btn3Motion	MotionNotify with button 3 down

Type	Meaning
Btn4Motion	MotionNotify with button 4 down
Btn5Motion	MotionNotify with button 5 down
Enter	EnterNotify
Leave	LeaveNotify
FocusIn	FocusIn
FocusOut	FocusOut
Keymap	KeymapNotify
Expose	Expose
GrExp	GraphicsExpose
NoExp	NoExpose
Visible	VisibilityNotify
Create	CreateNotify
Destroy	DestroyNotify
Unmap	UnmapNotify
Map	MapNotify
MapReq	MapRequest
Reparent	ReparentNotify
Configure	ConfigureNotify
ConfReq	ConfigureRequest
Grav	GravityNotify
ResReq	ResizeRequest
Circ	CirculateNotify
CircReq	CirculateRequest
Prop	PropertyNotify
SelClr	SelectionClear
SelReq	SelectionRequest
Select	SelectionNotify
Clrmap	ColormapNotify
Message	ClientMessage
Mapping	MappingNotify

Supported Abbreviations:

Abbreviation	Meaning
Ctrl	KeyPress with control modifier
Meta	KeyPress with meta modifier
Shift	KeyPress with shift modifier
Btn1Down	ButtonPress with Btn1 detail
Btn1Up	ButtonRelease with Btn1 detail
Btn2Down	ButtonPress with Btn2 detail
Btn2Up	ButtonRelease with Btn2 detail
Btn3Down	ButtonPress with Btn3 detail
Btn3Up	ButtonRelease with Btn3 detail
Btn4Down	ButtonPress with Btn4 detail
Btn4Up	ButtonRelease with Btn4 detail
Btn5Down	ButtonPress with Btn5 detail
Btn5Up	ButtonRelease with Btn5 detail

The Detail field is event specific and normally corresponds to the detail field of an X Event, for example, <Key>A. If no detail field is specified, then ANY is assumed.

Useful Examples

- Always put more specific events in the table before more general ones:

```
Shift <Btn1Down> : twas()\n\
<Btn1Down> : brillig()
```

- For double-click on Button 1 Up with Shift, use:

```
Shift<Btn1Up>(2) : and()
```

  This is equivalent to

```
Shift<Btn1Down>,Shift<Btn1Up>,Shift<Btn1Down>,Shift<Btn1Up> : and()
```

  with appropriate timers set between events.
- For double-click on Button 1 Down with Shift, use:

```
Shift<Btn1Down>(2) : the()
```

  This is equivalent to

```
Shift<Btn1Down>,Shift<Btn1Up>,Shift<Btn1Down> : the()
```

  with appropriate timers set between events.
- Mouse motion is always discarded when it occurs between events in a table where no motion event is specified:

```
<Btn1Down> <Btn1Up> : slithy()
```

  This is taken, even if the pointer jiggles a bit between the down and up events. Similarly, any motion event specified in a translation matches any number of motion events. If the motion event causes an action procedure to be invoked, the procedure is invoked after each motion event.
- If an event sequence consists of a sequence of events that is also a non-initial subsequence of another translation, it is not taken if it occurs in the context of the longer sequence. This occurs mostly in sequences like:

```
<Btn1Down> <Btn1Up> : toves()\n\
<Btn1Up> :  did()
```

  The second translation is taken only if the button release is not preceded by a button press or if there are intervening events between the press and the release. Be particularly aware of this when using the repeat notation, above, with buttons and keys because their expansion includes additional events, and when specifying motion events because they are implicitly included between any two other events.
- For single click on Button 1 Up with Shift and Meta, use:

```
Shift Meta <Btn1Down>, Shift Meta<Btn1Up>: gyre()
```

- The "+" notation allows you to say "for any number of clicks greater than or equal to count", such as:

```
Shift <Btn1Up>(2+) : and()
```

- To say EnterNotify with any modifiers, use:

```
<Enter> : gimble()
```

- To say EnterNotify with no modifiers, use:

```
None <Enter> : in()
```

- To say EnterNotify with Button 1 Down and Button 2 Up and don't care about the other modifiers, use:

```
Button1 ~Button2 <Enter> : the()
```

- To say EnterNotify with Button1 Down and Button2 Down exclusively, use:

```
! Button1 Button2 <Enter> : wabe()
```

It is never necessary to use ~ with !.

Appendix C

Conversion Notes

1. In the alpha release X Toolkit, each widget class implemented an Xt<*Widget*>Create (for example, XtLabelCreate) function, in which most of the code was identical from widget to widget. In this X Toolkit, a single generic XtCreateWidget performs most of the common work and then calls the initialize procedure implemented for the particular widget class.
2. Each composite widget class also implemented the procedures Xt<*Widget*>Add and an Xt<*Widget*>Delete (for example, XtButtonBoxAddButton and XtButtonBoxDeleteButton). In the beta release X Toolkit, the composite generic procedures XtManageChildren and XtUnmanageChildren perform error-checking and screening out of certain children. Then, they call the change_managed procedure implemented for the widget's composite class. If the widget's parent has not yet been realized, the call on the change_managed procedure is delayed until realization time.
3. The new X Toolkit can be used to implement old-style calls by defining one-line procedures or macros that invoke a generic routine. For example, you could define the macro XtCreateLabel: as the :

```
#define XtCreateLabel(name, parent, args, num_args) \
    ((LabelWidget) XtCreateWidget(name, labelWidgetClass,  parent, args, num_args))
```

Index

X Toolkit Widgets – C Language X Interface

X Window System

X Version 11, Release 2

Ralph R. Swick

Digital Equipment Corporation
External Research Group
MIT Project Athena

Terry Weissman

Digital Equipment Corporation
Western Software Laboratory

Table of Contents

Acknowledgments

The implementation of the Athena Widgets was the responsibility of Ralph Swick, Ron Newman (Project Athena), and Mark Ackerman (Project Athena). Additional contributions to their implementation was made by:

Rich Hyde (Digital WSL)
Terry Weissman (Digital WSL)
Mary Larson (Digital UEG)
Joel McCormack (Digital WSL)
Jeanne Rich (Digital WSL)
Charles Haynes (Digital WSL)
Loretta Guarino-Reid (Digital WSL)

The contributors to the X10 toolkit also deserve much of the credit for this work. The Athena Widgets borrow heavily on the their counterparts in the X10 toolkit. The design and implementation of the X10 toolkit were done by:

Terry Weissman (Digital WSL)
Smokey Wallace (Digital WSL)
Phil Karlton (Digital WSL)
Charles Haynes (Digital WSL)
Ram Rao (Digital UEG)
Mary Larson (Digital UEG)
Mike Gancarz (Digital UEG)
Kathleen Langone (Digital UEG)

Thanks go to Al Mento of Digital's UEG Documentation Group for formatting and generally improving this document and to Chris Peterson of Project Athena for testing the many versions of the code and reviewing this document.

Ralph R. Swick
Digital Equipment Corporation
External Research Group
MIT Project Athena

Chapter 1

X Toolkit Overview

The X Toolkit provides the base functionality necessary to build a variety of application environments. The X Toolkit is extensible and supportive of the independent development of new or extended components. This is accomplished by defining interfaces that mask implementation details from both application and component implementors. By following a small set of conventions, a programmer can extend the X Toolkit and have these extensions function with existing facilities.

The X Toolkit is a library package layered on top of the X Window System. This layer extends the basic abstractions provided by X and provides the next layer of functionality primarily by supplying:

- Mechanisms for interactions between and within user-interface components
- A cohesive set of sample widgets

In the X Toolkit, a widget is the combination of an X window or subwindow and its associated input and output semantics.

To the extent possible, the X Toolkit is policy-free. The application environment, not the X Toolkit, defines, implements, and enforces:

- Policy
- Consistency
- Style

Each individual widget implementation defines its own policy. The X Toolkit design allows for but does not necessarily encourage the free mixing of radically differing widget implementations.

Note

This document describes the ''Athena Widgets'' set. Related future documents may describe other widget sets as they are made available.

1.1. Introduction to the X Toolkit Library

The X Toolkit library provides tools that simplify the design of application user interfaces in the X Window System programming environment. It assists application programmers by providing a set of common underlying user-interface functions. It also lets widget programmers modify existing widgets or add new widgets. By using the X Toolkit library in their applications, programmers present a similar user interface across applications to all workstation users.

The X Toolkit consists of:

- A set of Intrinsic mechanisms for building widgets
- An architectural model for constructing widgets
- A sample interface (widget set) for programming

The Intrinsic mechanisms are intended for the widget programmer. The architectural model lets the widget programmer design new widgets by using the Intrinsics and by combining other widgets. The application interface layers built on top of the X Toolkit include a coordinated set of widgets and composition policies. Some of these widgets and policies are application domain-specific, while others are common across a number of application domains.

The X Toolkit library provides an architectural model that can accommodate different kinds of application interface layers. In addition, the supplied toolkit functions are:

- Functionally complete and policy-free
- Stylistically and functionally consistent with the X Window System primitives
- Portable across languages, computer architectures, and operating systems

The X Toolkit library also can implement one or more application interface layers to:

- Verify the toolkit architecture
- Provide a base set of widgets and composition policies that can be incorporated in other application interface layers
- Make the X Toolkit immediately usable by those application programmers who find that a supplied application interface layer meets their needs

The remainder of this chapter discusses the X Toolkit:

- Terminology
- Model
- Design principles and philosophy

1.2. Terminology

The following terms are used throughout this manual.

Application programmer

A programmer who uses X Toolkit widgets to produce an application user interface.

Child

A widget that is contained in another widget (parent).

Class

The general group that a specific object belongs to.

Client

A routine that uses a widget in an application or for composing another widget.

Full name

The name of a widget instance appended to the full name of its parent.

Instance

A specific widget object as opposed to a general widget class.

Method

The functions or procedures that a widget itself implements.

Name

The name that is specific to an instance of a widget for a given client.

Object

A software data abstraction consisting of private data and private and public routines that operate on the private data. Users of the abstraction can interact with the object only through calls to the object's public routines.

Parent

A widget that contains at least one other widgets (child). A parent widget is also known as a composite widget.

Resource

A named piece of data in a widget that can be set by a client, by an application, or by user defaults.

Superclass

A larger class of which a specific class is a member. All members of a class also are members of the superclass.

User

A person interacting with a workstation.

Widget

An object providing a user interface abstraction (for example, a Scrollbar widget).

Widget class

The general group that a specific widget belongs to, which is also known as the type of the widget.

Widget programmer

A programmer who adds new widgets to the X Toolkit.

1.3. Underlying Model

The underlying architectural model is based on the following premises:

Widgets are X windows

Every user-interface widget is contained in a unique X window. The X window ID for a widget is readily available from the widget ID, so standard Xlib window manipulation procedures can operate on widgets.

Information hiding

The data for every widget is private to the widget. That is, the data is neither directly accessible nor visible outside of the module implementing the widget. All program interaction with the widget is performed by a set of operations (methods) that are defined for the widget.

Widget semantics and widget layout geometry

Widget semantics are clearly separated from widget layout geometry. Widgets are concerned with implementing specific user-interface semantics. They have little say over issues such as their size or placement relative to other widget peers. Mechanisms are provided for associating geometric managers with widgets and for widgets to make suggestions about their own geometry.

1.4. Design Principles and Philosophy

The X Toolkit follows two design principles throughout, which cover languages and language bindings as well as widget IDs.

Languages and Language Bindings

The X Toolkit facilitates access from objective languages. However, the X Toolkit library is conveniently usable by application programs written in nonobjective languages. Procedural interface guidelines are required when the X Toolkit is used with nonobjective languages.

The guidelines for the procedural interfaces are:

- Strings are passed as null-terminated character arrays.
- Most other arrays are passed using two parameters: a size and a pointer to the first element. This also applies to argument lists passed to widgets.
- Most numeric arguments are passed by value.
- Structures as arguments are avoided, unless a method for building them is provided for languages without pointers. Pointers embedded in structures are allowed, but they should be avoided if an equivalent alternative is available.

- Pointers are not recommended as return arguments, unless they will never have to be dereferenced by the caller. If they need to be dereferenced, the caller should allocate storage and pass the address to the procedure to fill in.
- Procedures can be passed as parameters.
- The ownership of dynamically allocated storage is determined on a case-by-case basis. The application is also permitted to replace the standard memory allocation and freeing routines used by the library.

Widget IDs

All references to widgets are made using a unique identifier that is known as the widget ID. The widget ID is returned to the client by the XtCreateWidget function. From an application programmer's perspective, a widget ID is an opaque data type; no particular interpretation may be assigned to it. Given a widget ID, the corresponding X window ID, the Display and Screen structures, and other information are readily available by using Intrinsic functions.

From a widget programmer's perspective, the widget ID actually is a pointer to a data structure that is known as the widget instance record. Several parts of the data structure are common to all widget types, while other parts are unique to a particular widget type. The widget's private data that is associated with a particular widget instance normally is included directly in the widget instance record.

Chapter 2

Using Widgets

Widgets serve as the primary tools for building a user interface or application environment. The widget set consists of primitive widgets (for example, a command button) and composite widgets (for example, a Dialog widget).

The remaining chapters of this guide explain the widgets and the geometry managers that work together to provide a set of user-interface components. These user-interface components serve as a default interface for application programmers who do not want to implement their own widgets. In addition, they serve as examples or a starting point for those widget programmers who, using the Intrinsic mechanisms, want to implement alternative application programming interfaces.

The remainder of this chapter discusses the common features of the X Toolkit widgets.

2.1. Initializing the Toolkit

You must invoke the toolkit initialization function XtInitialize prior to invoking any other toolkit routines. XtInitialize opens the X server connection, parses standard parts of the command line, and creates an initial widget that is to serve as the root of a tree of widgets that will be created by this application.

```
Widget XtInitialize(shell_name, application_class, options, num_options, argc, argv)
    String shell_name;
    String application_class;
    XrmOptionDescRec options[];
    Cardinal num_options;
    Cardinal *argc;
    String argv[];
```

shell_name	Specifies the name of the application shell widget instance, which usually is something generic like ''main''.
application_class	Specifies the class name of this application, which usually is the generic name for all instances of this application. By convention, the class name is formed by reversing the case of the application's first two letters. For example, an application named ''xterm'' would have a class name of ''XTerm''.
options	Specifies how to parse the command line for any application-specific resources. The options argument is passed as a parameter to XrmParseCommand. For further information, see *Xlib – C Language X Interface.*
num_options	Specifies the number of entries in options list.
argc	Specifies a pointer to the number of command line parameters.
argv	Specifies the command line parameters.

For further information about this function, see the *X Toolkit Intrinsics – C Language X Interface.*

2.2. Creating a Widget

Creating a widget is a three-step process. First, the widget instance is allocated, and various instance-specific attributes are set by using XtCreateWidget. Second, the widget's parent is informed of the new child by using XtManageChild. Finally, X windows are created for the parent and all its children by using XtRealizeWidget and specifying the top-most widget. The first two steps can be combined by using XtCreateManagedWidget. In addition, XtRealizeWidget is automatically called when the child becomes managed if the parent already is realized.

To allocate and initialize a widget, use XtCreateWidget.

```
Widget XtCreateWidget(name, widget_class, parent, args, num_args)
    String name;
    WidgetClass widget_class;
    Widget parent;
    ArgList args;
    Cardinal num_args;
```

name	Specifies the instance name for the created widget that is used for retrieving widget resources.
widget_class	Specifies the widget class pointer for the created widget.
parent	Specifies the parent widget ID.
args	Specifies the argument list. The argument list is a variable-length list composed of name and value pairs that contain information pertaining to the specific widget instance being created. For further information, see ''Creating Argument Lists''.
num_args	Specifies the number of arguments in the argument list. When the num_args is zero, the argument list is never referenced.

When a widget instance is successfully created, the widget identifier is returned to the application. If an error is encountered, the XtError routine is invoked to inform the user of the error.

For further information, see the *X Toolkit Intrinsics – C Language X Interface.*

Common Arguments in the Widget Argument List

Although a widget can have unique arguments that it understands, all widgets have common arguments that provide some regularity of operation. The common arguments allow arbitrary widgets to be managed by higher-level components without regards to the individual widget type. All widgets ignore any argument that they do not understand.

The following resources are retrieved from the argument list or from the resource database by all X Toolkit widgets:

Name	Type	Default	Description
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	1	Width of the border in pixels
XtNdestroyCallback	XtCallbackList	NULL	Callback for XtDestroyWidget
XtNheight	int	widget-dependent	Height of the widget
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNsensitive	Boolean	True	Whether widget should receive input
XtNtranslations	TranslationTable	none	event-to-action translations
XtNwidth	int	widget-dependent	Width of the widget
XtNx	int	0	x coordinate within parent
XtNy	int	0	y coordinate within parent

The following additional resources are retrieved from the argument list or from the resource database by many X Toolkit widgets:

Name	Type	Default	Description
XtNcallback	XtCallbackList	NULL	Callback functions and client data
XtNforeground	Pixel	Black	Foreground color

2.3. Realizing a Widget

The X Toolkit provides the application programmer with a single function that:

- First, creates an X window for the widget and, if it is a composite widget, for each of its managed children.
- Then, maps each window onto the screen.

```
void XtRealizeWidget(w)
    Widget w;
```

w Specifies the widget.

For further information about this function, see the *X Toolkit Intrinsics – C Language X Interface.*

2.4. Standard Widget Manipulation Functions

After a widget has been created, a client can interact with that widget by calling:

- One of the standard widget manipulation routines that provide functions that all widgets support
- A widget class-specific manipulation routine

The X Toolkit provides generic routines to provide the application programmer access to a set of standard widget functions. These routines let an application or composite widget manipulate widgets without requiring explicit knowledge of the widget type. The standard widget manipulation functions let you:

- Control the location, size and mapping of widget windows
- Control the screen updating process
- Destroy a widget instance
- Obtain an argument value
- Set an argument value

Mapping Widgets

By default, widget windows automatically are mapped (made viewable) by XtRealizeWidget. This behavior can be changed by using XtSetMappedWhenManaged, and it then is the client's responsibility to use the XtMapWidget function to make the widget viewable.

```
void XtSetMappedWhenManaged(w, map_when_managed)
    Widget w;
    Boolean map_when_managed;
```

w Specifies the widget.

map_when_managed
Specifies the new value. If map_when_managed is True, the widget is mapped automatically when it is realized. If map_when_managed is False, the client must call XtMapWidget or make a second call to XtSetMappedWhenManaged to cause the child window to be mapped.

The definition for XtMapWidget is:

```
XtMapWidget(w)
    Widget w;
```

w Specifies the widget.

When you create several children in sequence for a common parent after it has been realized, it generally is more efficient to construct a list of children as they are created and use XtManageChildren to inform their parent of them all at once, instead of causing each child to be managed separately. By managing a list of children at one time, the parent can avoid wasteful duplication of geometry processing.

```
void XtManageChildren(children, num_children)
    WidgetList children;
    Cardinal num_children;
```

children Specifies a list of children to add.

num_children Specifies the number of children to add.

If the parent already is visible on the screen, it is especially important to batch updates so that the minimum amount of visible window reconfiguration is performed.

For further information about these functions, see the *X Toolkit Intrinsics – C Language X Interface.*

Destroying Widgets

To destroy a widget instance of any type, use XtDestroyWidget.

```
void XtDestroyWidget(w)
    Widget w;
```

w	Specifies the widget.

XtDestroyWidget destroys the widget and recursively destroys any children that it may have, including the windows created by its children. After calling XtDestroyWidget, no further references should be made to the widget or to the widget IDs of any children that the destroyed widget may have had.

Retrieving Widget Resource Values

To retrieve the current value of a resource attribute associated with a widget instance, use XtGetValues.

```
void XtGetValues(w, args, num_args)
    Widget w;
    ArgList args;
    Cardinal num_args;
```

w	Specifies the widget.
args	Specifies a variable length argument list of name/address pairs which contain the resource name and the address to store the resource value into. The arguments and values passed in args are dependent on the widget. The storage for argument values which are pointed to by args must be allocated and deallocated by the caller.
num_args	Specifies the number of arguments in argument list.

The arguments and values passed in the argument list are dependent upon the widget.

Modifying Widget Resource Values

To modify the current value of a resource attribute associated with a widget instance, use XtSetValues.

```
void XtSetValues(w, args, num_args)
    Widget w;
    ArgList args;
    Cardinal num_args;
```

w	Specifies the widget.
args	Specifies a variable length argument list of name/value pairs which contain the arguments to be modified and their new values. The arguments and values passed are dependent on the widget being modified.
num_args	Specifies the number of arguments in the argument list.

The arguments and values passed in the argument list depend upon the widget being modified. Some widgets may not allow certain resources to be modified after the widget instance has been created or realized. No notification is given if any part of a XtSetValues request is ignored.

For further information about these functions, see the *X Toolkit Intrinsics – C Language X Interface*.

Note

The argument list entry for XtGetValues specifies the address to which the caller wants the value copied. The argument list entry for XtSetValues, however, contains the new value itself if the value is four bytes or less; otherwise, it is a pointer to the value.

2.5. Using the Client Callback Interface

Widgets communicate changes in their state to their clients by means of a callback facility. The format for a client's callback handler is:

```
void CallbackProc(w, client_data, call_data);
    Widget w;
    caddr_t client_data;
    caddr_t call_data;
```

w	Specifies widget for which the callback is registered.
client_data	Specifies the arbitrary client-supplied data that the widget should pass back to the client when the widget executes the client's callback procedure. This is a way for the client registering the callback to also register client-specific data: a pointer to additional information about the widget, a reason for invoking the callback, and so on. It is perfectly normal to have client_data of NULL if all necessary information is in the widget.
call_data	Specifies any callback-specific data the widget wants to pass to the client. For example, when Scrollbar executes its thumbProc callback list, it passes the current position of the thumb in the call_data argument.

Callbacks can be registered with widgets in one of two ways. When the widget is created, a pointer to a list of callback procedure and data pairs can be passed in the argument list to XtCreateWidget. The list is of type XtCallbackList:

```
typedef struct {
        XtCallbackProc callback;
        caddr_t client_data;
} XtCallbackRec, XtCallbackList;
```

The callback list must be allocated and initialized prior to calling XtCreateWidget. Once the widget is created, the client can change or de-allocate this list; The widget, itself, makes no further reference to it.

The second method for registering callbacks is to use XtAddCallback after the widget has been created.

```
void XtAddCallback(w, callback_name, callback, client_data)
    Widget w;
    String callback_name;
    XtCallbackProc callback;
    caddr_t client_data;
```

w	Specifies the widget to add the callback to.
callback_name	Specifies the callback list within the widget to append to.
callback	Specifies the callback procedure to add.
client_data	Specifies the data to be passed to the callback when it is invoked.

XtAddCallback adds the specified callback to the list for the named widget.

All widgets provide a callback list named XtNdestroyCallback where clients can register procedures that are to be executed when the widget is destroyed. The destroy callbacks are executed when the widget or an ancestor is destroyed. The call_data argument is unused for destroy callbacks.

The X Toolkit Intrinsics provide additional functions for further manipulating a callback list. For information about these function, see XtCallCallbacks, XtRemoveCallback, XtRemoveCallbacks, and XtRemoveAllCallbacks. in the *X Toolkit Intrinsics – C Language X Interface.*

2.6. Programming Considerations

This section provides some guidelines to set up an application program that uses the X Toolkit.

Writing Applications

When writing an application that uses the toolkit, the application needs to perform a standard set of steps:

1. Include <X11/Intrinsic.h> in your application programs. This header file automatically includes <X11/Xlib.h>, so all Xlib functions also are defined.
2. Include the widget-specific header files for each widget type that you need to use. For example, <X11/Label.h> and <X11/Command.h>.
3. Call the XtInitialize function prior to invoking any other toolkit or Xlib functions. For further information, see Section 2.1 and the *X Toolkit Intrinsics – C Language X Interface.*
4. To pass attributes to the widget creation routines, set up argument lists. In this document, a list of valid argument names that start with XtN is provided in the discussion of each widget.

   For example, while creating a Command button, an application may provide a label string to the Command button, such as ''PressSpacesToExit''. The application also should provide a function name that the application calls when the command button is selected.

   For further information, see ''Creating Argument Lists''.
5. When the argument list is set up, create the widget by using the XtCreateWidget function. For further information, see Section 2.2 and the *X Toolkit Intrinsics – C Language X Interface.*

6. If the widget has any callback routines, which are usually defined by using the XtNcallback argument or the XtAddCallback function, declare these routines within the application.
7. After a widget has been created, the widget should be managed by using XtManageChild. If there is no manipulation of the widget between XtCreateWidget and XtManageChild, you can do this in a single step by using XtCreateManagedWidget. For further information about these functions, see the *X Toolkit Intrinsics – C Language X Interface.*
8. Most applications now sit in a loop processing events using XtMainLoop, for example:

   ```
   XtCreateManagedWidget(name, class, parent, args, num_args);
   XtRealizeWidget(parent);
   XtMainLoop();
   ```

   For information about this function, see the *X Toolkit Intrinsics – C Language X Interface.*
9. Link your application with libXaw.a (the Athena widgets), libXt.a (the X Toolkit Intrinsics), and libX.a (the core X library). The following provides a sample command line:

   ```
   cc -o application application.c –lXaw –lXt –lX
   ```

Creating Argument Lists

To set up an argument list for the inline specification of widget attributes, you can use one of the three approaches discussed in this section. You should use whichever approach fits the needs of the application and you are most comfortable with. In general, argument lists should be kept as short as possible to allow widget attributes to be specified through the resource database. Whenever a client inserts a specific attribute value in an argument list, the user is prevented from customizing the behavior of the widget. Resource names in the resource database, by convention, correspond to their symbolic names that are used in argument list without the XtN prefix. For example, the resource name for XtNforeground is "foreground". For further information, see the *X Toolkit Intrinsics – C Language X Interface.*

The Arg structure contains:

```
typedef struct {
        String name;
        XtArgVal value;
} Arg, *ArgList;
```

Approach 1

You can statically initialize the argument list, as in this example:

```
static Arg arglist[] = {
        {XtNwidth, (XtArgVal) 400},
        {XtNheight, (XtArgVal) 300},
};
```

This approach makes it easy to add or delete new elements. The XtNumber macro can be used to compute the number of elements in the argument list, thus preventing simple programming errors. The following provides an example:

```
XtCreateWidget(name, class, parent, args, XtNumber(args));
```

Approach 2

You can use the XtSetArg macro:

```
Arg arglist[10];
XtSetArg(arglist[1], XtNwidth, 400);
XtSetArg(arglist[2], XtNheight, 300);
```

To make it easier to insert and delete entries, you also can use a variable index, as in this example:

```
Arg arglist[10];
Cardinal i=0;
XtSetArg(arglist[i], XtNwidth,  400);     i++;
XtSetArg(arglist[i], XtNheight, 300);     i++;
```

The i variable can then be used as the argument list count in the widget create function. In this example, XtNumber would return 10, not 2, and, therefore, is not useful.

Note

You should not use auto-increment or auto-decrement within the first argument to XtSetArg. As it is currently implemented, XtSetArg is a macro that dereferences the first argument twice.

Approach 3

You can individually set the elements of the argument list array, one piece at a time:

```
Arg arglist[10];
arglist[0].name  = XtNwidth;
arglist[0].value = (XtArgVal) 400;
arglist[1].name  = XtNheight;
arglist[1].value = (XtArgVal) 300;
```

Also in this example, XtNumber would return 10, not 3, and, therefore, is not useful.

Approach 4

Use a mixture of Approaches 1 and 3. The argument list is statically defined but some entries are modified at run-time.

```
static Arg arglist[] = {
        {XtNwidth, (XtArgVal) 400},
        {XtNheight, (XtArgVal) NULL},
};
arglist[1].value = (XtArgVal) 300;
```

In this example, XtNumber can be used, as in Approach 1, for easier code maintenance.

Sample Program

This sample program creates one command button that, when pressed, causes the program to exit. This example is a complete program that illustrates:

- Toolkit initialization
- Optional command-line arguments
- Widget creation
- Callback routines

command widget header

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Command.h>

static XrmOptionDescRec options[] = {
{"-label", "*button.label", XrmoptionSepArg, NULL}
};

Syntax(call)
        char *call;
{
        fprintf(stderr, "Usage: %s\n", call);
}

void Activate(w, client_data, call_data)
        Widget w;
        caddr_t client_data;
        caddr_t call_data;
{
        printf("button was activated.\n");
        exit(0);  <- exits program
}

void main(argc, argv)
        unsigned int argc;
        char **argv;
{
        Widget toplevel;
        static XtCallbackRec callbacks[] = {
          { Activate, NULL },
          { NULL, NULL },
        };
                            type
                            cast
        static Arg args[] = {
         { XtNcallback, (XtArgVal)callbacks },
        };

        toplevel = XtInitialize("main", "Demo", options, XtNumber(options), &argc, argv );
        if (argc != 1) Syntax(argv[0]);

        XtCreateManagedWidget("button",commandWidgetClass,toplevel,args,XtNumber(args));

        XtRealizeWidget(toplevel);             widget         widget
        XtMainLoop();                          class          instantiation
}
```

Chapter 3

Widgets

This chapter describes the following X Toolkit widgets:

- Command buttons
- Labels
- Text
- Scrollbars
- Viewports
- Boxes
- VPaned
- Forms
- Dialog
- Grip

3.1. Command Button Widget

The Command button widget is a rectangle that contains a text label. When the pointer cursor is on the button, its border is highlighted to indicate that the button is available for selection. Then, when a pointer button is clicked, the button is selected, and the application's callback routine is invoked.

The class variable for the Command button widget is commandWidgetClass.

When creating a Command button widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	1	Width of button border
XtNcallback	XtCallbackList	NULL	Callback for button select
XtNcursor	Cursor	opendot	Pointer cursor
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for XtDestroyWidget
XtNfont	FontInfo*	fixed	Label font
XtNforeground	Pixel	Black	Foreground color
XtNheight	int	text height	Button height
XtNhighlightThickness	int	2	Width of border to be highlighted
XtNinsensitiveBorder	Pixmap	Gray	Border when not sensitive
XtNinternalHeight	int	2	Internal border height for highlighting
XtNinternalWidth	int	4	Internal border width for highlighting
XtNjustify	XtJustify	XtjustifyCenter	Type of text alignment
XtNlabel	char*	Button name	Button label
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNsensitive	Boolean	True	Whether widget receives input
XtNtranslations	TranslationTable	none	event-to-action translations

Name	Type	Default	Description
XtNwidth	int	text width	Button width
XtNx	int	0	Widget x coordinate
XtNy	int	0	Widget y coordinate

XtNheight	Specifies the height of the Command widget. The default value is the minimum height that will contain: XtNinternalheight + height of XtNlabel + XtNinternalHeight If the specified height is larger than the minimum, the label string is centered vertically.
XtNinternalHeight	Represents the distance in pixels between the top and bottom of the label text and the horizontal edges of the Command widget. HighlightThickness can be larger or smaller than this value.
XtNinternalWidth	Represents the distance in pixels between the ends of the label text and the outer vertical edges of the Command widget. HighlightThickness can be larger or smaller than this value.
XtNjustify	Specifies left, center, or right alignment of the label string within the Command widget. If it is specified within an ArgList, one of the values XtJustifyLeft, XtJustifyCenter, or XtJustifyRight can be specified. In a resource of type "string", one of the values "left", "center", or "right" can be specified.
XtNlabel	Specifies the text string that is to be displayed in the Command widget. The default is the widget name of the Command widget.
XtNsensitive	Causes the Command widget to display a different border and a stippled label string when the Command widget is set to insensitive.
XtNwidth	Specifies the width of the Command widget. The default value is the minimum width that will contain: XtNinternalWidth + width of XtNlabel + XtNinternalWidth If the width is larger or smaller than the minimum, XtNjustify determines how the label string is aligned.

The Command button widget supports the following actions:

- Switching the button between the foreground and background colors with set and unset
- Processing application callbacks with notify
- Switching the internal border between highlighted and unhighlighted states with highlight and unhighlight

The following are the default translation bindings that are used by the Command button widget:

```
<BtnDown>ButtonDown1: set()
<BtnUp>ButtonDown1:   notify() unset()
<EnterWindow>:        highlight()
<LeaveWindow>:        unset() unhighlight()
```

To create a Command button widget instance, use XtCreateWidget and specify the class variable commandWidgetClass.

To destroy a Command button widget instance, use XtDestroyWidget and specify the widget ID of the button.

The Command button widget supports two callback lists: XtNdestroyCallback and XtNcallback. The notify action executes the callbacks on the the XtNcallback list. The call_data argument is unused.

3.2. Label Widget

A Label is an uneditable text string that is displayed within a window. The string is limited to one line and can be aligned to the left, right, or center of its window. A Label can neither be selected nor edited by the user.

The class variable for the Label widget is labelWidgetClass.

When creating a Label widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	1	Border width in pixels
XtNcursor	Cursor	None	Pointer cursor
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for XtDestroyWidget
XtNfont	FontInfo*	fixed	Label font
XtNforeground	Pixel	Black	Foreground color
XtNheight	int	text height	Height of widget
XtNinsensitiveBorder	Pixmap	Gray	Border when not sensitive
XtNinternalHeight	int	2	See note
XtNinternalWidth	int	4	See note
XtNjustify	XtJustify	XtjustifyCenter	Type of text alignment
XtNlabel	char*	label name	String to be displayed
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNsensitive	Boolean	True	Whether widget receives input
XtNwidth	int	text width	Width of widget
XtNx	int	0	x coordinate in pixels
XtNy	int	0	y coordinate in pixels

XtNheight	Specifies the height of the Label widget. The default value is the minimum height that will contain: XtNinternalheight + height of XtNlabel + XtNinternalHeight If the specified height is larger than the minimum, the label string is centered vertically.
XtNinternalHeight	Represents the distance in pixels between the top and bottom of the label text and the horizontal edges of the Label widget.
XtNinternalWidth	Represents the distance in pixels between the ends of the label text and the outer vertical edges of the Label widget.

XtNjustify	Specifies left, center, or right alignment of the label string within the Label widget. If it is specified within an ArgList, one of the values XtJustifyLeft, XtJustifyCenter, or XtJustifyRight can be specified. In a resource of type ''string'', one of the values ''left'', ''center'', or ''right'' can be specified.
XtNlabel	Specifies the text string that is to be displayed in the button. The default is the widget name of the Label widget.
XtNsensitive	Causes the button to display a different border and a stippled label string when the Label widget is set to insensitive.
XtNwidth	Specifies the width of the Label widget. The default value is the minimum width that will contain: XtNinternalWidth + width of XtNlabel + XtNinternalWidth If the width is larger or smaller than the minimum, XtNjustify determines how the label string is aligned.

To create a Label widget instance, use XtCreateWidget and specify the class variable labelWidgetClass.

To destroy a Label widget instance, use XtDestroyWidget and specify the widget ID of the label.

The Label widget supports only the XtNdestroyCallback callback list.

3.3. Text Widget

A Text widget is a window that provides a way for an application to display one or more lines of text. The displayed text can reside in a file on disk or in a string in memory. An option also lets an application display a vertical Scrollbar in the Text window, letting the user scroll through the displayed text. Other options allow an application to let the user modify the text in the window.

The Text widget is divided into three parts:

- Source
- Sink
- Text widget

The idea is to separate the storage of the text (source) from the painting of the text (sink). The Text widget coordinates the sources and sinks. Clients normally will use AsciiText widgets that automatically create the source and sink for the client. A client can, if it so chooses, explicitly create the source and sink before creating the Text widget.

The source stores and manipulates the text. The X Toolkit provides string and disk file sources. The source determines what functions may be performed on the text.

The sink obtains the fonts and the colors in which to paint the text. The sink also computes what text can fit on each line. The X Toolkit provides a single font, single color ASCII sink.

If a disk file is used to display the text, two edit modes are available:

- Append
- Read-only

Append mode lets the user enter text into the window, while read-only mode does not. As the text is entered, it is added after the last character in the window, regardless of where the user positions the cursor.

If a string in memory is used, the application must allocate the amount of space needed. If a string in memory is used to display text, three types of edit mode are available:

- Append-only
- Read-only
- Editable

The first two modes are the same as displaying text from a disk file. Editable mode lets the user place the cursor anywhere in the text and modify the text at that position. The text cursor position can be modified by using the key strokes or pointer buttons defined by the event bindings. All standard keyboard editing facilities are supported by the event bindings. The following actions are supported:

Cursor Movement
- forward-character
- backward-character
- forward-word
- backward-word
- forward-paragraph
- backward-paragraph
- beginning-of-line
- end-of-line
- next-line
- previous-line
- next-page
- previous-page
- beginning-of-file
- end-of-file
- scroll-one-line-up
- scroll-one-line-down

Delete
- delete-next-character
- delete-previous-character
- delete-next-word
- delete-previous-word
- delete-selection

Selection
- select-word
- select-all
- select-start
- select-adjust
- select-end
- extend-start
- extend-adjust
- extend-end

New Line
- newline-and-indent
- newline-and-backup
- newline

Miscellaneous
- redraw-display
- insert-file
- do-nothing

Kill
- kill-word
- backward-kill-word
- kill-selection
- kill-to-end-of-line
- kill-to-end-of-paragraph

Unkill
- unkill
- stuff

Notes

1. A page corresponds to the size of the Text window. For example, if the Text window is 50 lines in length, scrolling forward one page is the same as scrolling forward 50 lines.
2. The delete binding deletes a text item. The kill binding deletes a text item and puts the item in the kill buffer (X buffer 1).

3. The unkill binding inserts the contents of the kill buffer into the text at the current position. The stuff binding inserts the contents of the select buffer (X buffer 0) into the text at the current position.

The default event bindings for the Text widget are:

```
char *defaultTextEventBindings[] = {
        "Ctrl<Key>F:            forward-character( )\n\
        <Key>Right:             forward-character( )\n\
        Ctrl<Key>B:             backward-character( )\n\
        <Key>Left:              backward-character( )\n\
        Meta<Key>F:             forward-word( )\n\
        Meta<Key>B:             backward-word( )\n\
        Meta<Key>]:             forward-paragraph( )\n\
        Ctrl<Key>[:             backward-paragraph( )\n\
        Ctrl<Key>A:             beginning-of-line( )\n\
        Ctrl<Key>E:             end-of-line( )\n\
        Ctrl<Key>N:             next-line( )\n\
        <Key>Down:              next-line( )\n\
        Ctrl<Key>P:             previous-line( )\n\
        <Key>Up:                previous-line( )\n\
        Ctrl<Key>V:             next-page( )\n\
        Meta<Key>V:             previous-page( )\n\
        Meta<Key>\<:            beginning-of-file( )\n\
        Meta<Key>\>:            end-of-file( )\n\
        Ctrl<Key>Z:             scroll-one-line-up( )\n\
        Meta<Key>Z:             scroll-one-line-down( )\n\
        Ctrl<Key>D:             delete-next-character( )\n\
        Ctrl<Key>H:             delete-previous-character( )\n\
        <Key>Num_Lock:          delete-previous-character( )\n\
        <Key>Delete:            delete-previous-character( )\n\
        <Key>BackSpace:         delete-previous-character( )\n\
        Meta<Key>D:             delete-next-word( )\n\
        Meta<Key>H:             delete-previous-word( )\n\
        Shift Meta<Key>D:       kill-word( )\n\
        Shift Meta<Key>H:       backward-kill-word( )\n\
        Ctrl<Key>W:             kill-selection( )\n\
        Ctrl<Key>K:             kill-to-end-of-line( )\n\
        Meta<Key>K:             kill-to-end-of-paragraph( )\n\
        Ctrl<Key>Y:             unkill( )\n\
        Meta<Key>Y:             stuff( )\n\
        Ctrl<Key>J:             newline-and-indent( )\n\
        <Key>Linefeed:          newline-and-indent( )\n\
        Ctrl<Key>O:             newline-and-backup( )\n\
        Ctrl<Key>M:             newline( )\n\
        <Key>Return:            newline( )\n\
        Ctrl<Key>L:             redraw-display( )\n\
        Meta<Key>I:             insert-file( )\n\
        <FocusIn>:              focus-in( )\n\
        <FocusOut>:             focus-out( )\n\
<Btn1Down>:             select-start( )\n\
Button1<PtrMoved>:      extend-adjust( )\n\
<Btn1Up>:               extend-end( )\n\
<Btn2Down>:             stuff( )\n\
<Btn3Down>:             extend-start( )\n\
Button3<PtrMoved>:      extend-adjust( )\n\
<Btn3Up>:               extend-end( )\n\
<Key>:                  insert-char( )\n\
Shift<Key>:             insert-char( )";
```

A user-supplied resource entry can use application-specific bindings, a subset of the supplied default bindings, or both. The following is an example of a user-supplied resource entry that uses a subset of the default bindings:

```
xmh*Text.translation:\n
        <Key>Right:     forward-character( )\n
        <Key>Left:      backward-character( )\n
        <Meta>F:        forward-word( )\n
        <Meta>B:        backward-word( )\n
        <Meta>]:        forward-paragraph( )\n
        <Ctrl>[:        backward-paragraph( )
```

A Text widget lets both the user and the application take control of the text being displayed. The user takes control with the scroll bar or with key strokes defined by the event bindings. The scroll bar option places the scroll bar on the left side of the window and can be used with any editing mode. The application takes control with procedure calls to the Text widget to:

- Display text at a specified position
- Highlight specified text areas
- Replace specified text areas

The text that is selected within a Text window is put into buffer 0 (zero) and can be retrieved by the application with the Xlib XFetchBytes function. All standard selection schemes are supported by the event bindings.

The class variable for the Text widget is textWidgetClass.

To create a Text string widget, use XtCreateWidget and specify the class variable asciiStringWidgetClass.

To create a Text file widget, use XtCreateWidget and specify the class variable asciiDiskWidgetClass.

Note

If you want to create an instance of the class textWidgetClass, you must provide a source and a sink when the widget is created. The Text widget cannot be instantiated without either or both.

When creating a Text widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap

Name	Type	Default	Description
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	4	Border width in pixels
XtNcursor	Cursor	XC_xterm	Pointer cursor
XtNdialogHOffset	int	10	Offset of insert file dialog
XtNdialogVOffset	int	10	Offset of insert file dialog
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for XtDestroyWidget
XtNdisplayPosition	int	0	Byte offset of first line
XtNeditType	XtEditType	XttextRead	Edit mode (see note)
XtNfile	char*	tmpnam()	File name for asciiDiskWidgetClass
XtNforeground	Pixel	Black	Foreground color
XtNfont	FontInfo*	fixed	Fontname
XtNheight	int	font height	Height of widget
XtNinsertPosition	int	0	Byte offset of cursor position
XtNleftMargin	int	2	Left margin in pixels
XtNlength	int	string length	Length for asciiStringWidgetClass
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNmaxLength	int	1024	Maximum string size for asciiStringWidgetClass
XtNselectTypes	XtTextSelectType*	See below	See below
XtNselection	XtTextBlock	none	String that is highlighted
XtNsensitive	Boolean	True	Whether widget receives input
XtNstring	char*	blank	String for asciiStringWidgetClass
XtNtextOptions	int	none	See below
XtNtextSink	XtTextSink	none	See below
XtNtextSource	XtTextSource	none	See below
XtNtranslations	TranslationTable	none	event-to-action translations
XtNwidth	int	100	Width of widget (pixels)
XtNx	int	0	x coordinate in pixels
XtNy	int	0	y coordinate in pixels

Notes

1. You cannot use XtNeditType, XtNfile, XtNlength, and XtNfont with the XtTextSetValues and the XtTextGetValues calls.
2. The XtNeditType attribute has one of the values XttextAppend, XttextEdit, or XttextRead.

The options for the XtNtextOptions attribute are:

Option	Description
editable	Whether or not the user is allowed to modify the text.
resizeHeight	Makes a request to the parent widget to lengthen the window if all the text cannot fit in the window.
resizeWidth	Makes a request to the parent widget to widen the window if the text becomes too long to fit on one line.
scrollHorizontal	Puts a scroll bar on the top of the window.
scrollOnOverflow	Automatically scrolls the text up when new text is entered below the bottom (last) line.

Option	Description
scrollVertical	Puts a scroll bar on the left side of the window.
wordBreak	Starts newline when word does not fit on current line.

These options can be ORed together to set more than one at the same time. XtNselectionTypes is an array of entries of type XtTextSelectType and is used for multiclick. As the pointer button is clicked in rapid succession, each click highlights the next "type" described in the array.

XtselectAll	Selects the contents of the entire buffer.
XtselectChar	Selects text characters as the pointer moves over them.
XtselectLine	Selects the entire line.
XtselectNull	Indicates the end of the selection array.
XtselectParagraph	Selects the entire paragraph (delimited by newline characters).
XtselectPosition	Selects the current pointer position.
XtselectWord	Selects whole words (delimited by whitespace) as the pointer moves onto them.

The default selectType array is:

{XtselectPosition, XtselectWord, XtselectLine, XtselectParagraph, XtselectAll, XtselectNull}

For the default case, two rapid pointer clicks highlight the current word, three clicks highlight the current line, four clicks highlight the current paragraph, and five clicks highlight the entire text. If the timeout value is exceeded, the next pointer click returns to the first entry in the selection array. The selection array is not copied by the Text widget. The client must allocate space for the array and may not deallocate or change it until the Text widget is destroyed or until a new selection array is set.

Selecting Text

To enable an application to select a piece of text, use XtTextSetSelection.

```
typedef long XtTextPosition;

void XtTextSetSelection(w, left, right)
    Widget w;
    XtTextPosition left, right;
```

w	Specifies the window ID.
left	Specifies the byte offset at which the the selection begins.
right	Specifies the byte offset at which the selection ends.

This function highlights the text and puts the text into buffer 0 (zero). The text can be retrieved by using the Xlib XFetchBytes call.

Unhighlighting Text

To unhighlight previously highlighted text in a window, use XtTextUnsetSelection.

```
void XtTextUnsetSelection(w)
    Widget w;
```

Getting Selected Text Byte Positions

To enable the application to get the byte positions of the selected text, use XtTextGetSelection-Pos.

```
void XtTextGetSelectionPos(w, pos1, pos2)
    Widget w;
    XtTextPosition *pos1, *pos2;
```

w	Specifies the window ID.
pos1	Specifies an argument to which the beginning byte offset of the selection is returned.
pos2	Specifies an argument to which the ending byte offset of the selection is returned.

Replacing Text

To enable an application to replace text, use XtTextReplace.

```
int XtTextReplace(w, start_pos, end_pos, text)
    Widget w;
    XtTextPosition start_pos, end_pos;
    XtTextBlock *text;
```

w	Specifies the window ID.
start_pos	Specifies the starting byte position of the text replacement.
end_pos	Specifies the ending byte position of the text replacement.
text	Specifies the text to be inserted into the file.

The XtTextBlock structure contains:

```
typedef struct {
        int firstPos;
        int length;
        char *ptr;
} XtTextBlock, *TextBlockPtr;
```

The firstPos field is the starting point to use within the ptr field. The value is usually zero. The length field is the number of bytes that are transferred from the ptr field. The number of bytes transferred is usually the number of bytes in ptr. The XtTextReplace arguments start_pos and end_pos represent the text source byte offsets for text that is found in the XtTextBlock structure. The characters between start_pos and end_pos are deleted, and the characters that are specified by text are inserted in their place.

The XtTextReplace function has no effect on a read-only, disk-source Text widget.

Note

Only ASCII text currently is supported, and only one font may be used for each Text widget.

3.4. Scrollbar Widget

The Scrollbar widget is a rectangular box that contains a slide region and a thumb (slide bar). A Scrollbar can be used alone, as a valuator, or it can be used within a composite widget (for example, a Viewport). A Scrollbar can be aligned either vertically or horizontally.

When a Scrollbar is created, it is drawn with the thumb in a contrasting color. The thumb is used to scroll the data and to give visual feedback on the percentage of the data visible.

Each pointer button invokes a specific scrollbar action. That is, given either a vertical or horizontal alignment, the pointer button actions will scroll or return data as appropriate for that alignment. Pointer button 1 and button 3 do not perform scrolling operations by default. Instead, they return the pixel position of the cursor on the scroll region. When pointer button 2 is clicked, the thumb moves to the current pointer position. When pointer button 2 is held down and the pointer pointer is moved, the thumb moves in the appropriate direction.

The cursor in the scroll region changes depending on the current action. When no pointer button is pressed, the cursor appears as an arrow that points in the direction that scrolling can occur. When either pointer button 1 or button 2 is pressed, the cursor appears as a single-headed arrow that points in the direction that scrolling can occur. When pointer button 2 is pressed, the cursor appears as an arrow that points to the thumb.

While scrolling is in progress, the application receives notification from callback procedures. For both scrolling actions, the callback returns the Scrollbar widget ID, the client_data, and the pixel position relative to the current thumb position. For smooth scrolling, the callback routine returns the scroll bar window, the client data, and the current relative position of the thumb. When the thumb is moved using pointer button 2, the callback procedure is invoked continuously. When the thumb is moved using either button 1 or button 2, the callback procedure is invoked only when the button is released.

The class variable for the Scrollbar widget is scrollbarWidgetClass.

When creating a Scrollbar widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	1	Width of button border
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for XtDestroyWidget
XtNforeground	Pixel	Black	Thumb color
XtNheight	int	See below	Height of scroll bar
XtNlength	int	None	major dimension (height of XtorientVertical)
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNorientation	XtOrientation	XtorientVertical	Orientation (vertical or horizontal)
XtNscrollDCursor	Cursor	XC_sb_down_arrow	Cursor for scrolling down
XtNscrollHCursor	Cursor	XC_sb_h_double_arrow	Idle horizontal cursor
XtNscrollLCursor	Cursor	XC_sb_left_arrow	Cursor for scrolling left
XtNscrollProc	XtCallbackList	NULL	Callback for the slide region

Name	Type	Default	Description
XtNscrollRCursor	Cursor	XC_sb_right_arrow	Cursor for scrolling right
XtNscrollUCursor	Cursor	XC_sb_up_arrow	Cursor for scrolling up
XtNscrollVCursor	Cursor	XC_sb_v_double_arrow	Idle vertical cursor
XtNsensitive	Boolean	True	Whether widget receives input
XtNshown	float	NULL	Percentage the thumb covers
XtNthickness	int	14	minor dimension (height if XtorientHorizontal)
XtNthumb	Pixmap	Grey	Thump pixmap
XtNthumbProc	XtCallbackList	NULL	Callback for thumb select
XtNtop	float	NULL	Position on scroll bar
XtNtranslations	TranslationTable	none	event-to-action translations
XtNwidth	int	See below	Width of scroll bar
XtNx	int	NULL	x position of scroll bar
XtNy	int	NULL	y position of scroll bar

You can set the dimensions of the Scrollbar two ways. As for all widgets, you can use the XtNwidth and XtNheight resources. In addition, you can use an alternative method that is independent of the vertical or horizontal orientation:

XtNlength	Specifies the height for a vertical Scrollbar and the width for a horizontal Scrollbar.
XtNthickness	Specifies the width for a vertical Scrollbar and the height for a horizontal Scrollbar.

To create a Scrollbar widget instance, use XtCreateWidget and specify the class variable scrollbarWidgetClass.

The callback procedure for the slide area is:

```
void ScrollProc(scrollbar, client_data, position)
    Widget scrollbar;
    caddr_t client_data;
    int position;
```

scrollbar	Specifies the ID of the Scrollbar.
client_data	Specifies the client data.
position	Returns the pixel position of the thumb in integer form.

The ScrollProc callback is used for incremental scrolling and for getting feedback from the thumb and background. The position argument is a signed quantity. Using the default button bindings, Button 1 returns a positive value, and Button 3 returns a negative value. In both cases, the magnitude of the value is the distance of the pointer from the top (or left) of the Scrollbar. The value will never be less than zero or greater than the length of the Scrollbar.

The callback procedure for the thumb area is:

```
void ThumbProc(scrollbar, client_data, percent)
    Widget scrollbar;
    caddr_t client_data;
    float percent;
```

scrollbar	Specifies the ID of the scroll bar window.

client_data	Specifies the client data.
percent	Specifies the floating point position of the thumb (0.0 – 1.0).

The ThumbProc callback is used to implement smooth scrolling. With the default button bindings, Button2 moves the thumb interactively, and the ThumbProc is called on each new position of the pointer.

B

To set the values of a Scrollbar thumb, use XtScrollbarSetThumb.

```
void XtScrollbarSetThumb(w, top, shown)
    Widget w;
    float top;
    float shown;
```

w	Specifies the Scrollbar widget ID.
top	Specifies the position of the top of the thumb as a fraction of the length of the Scrollbar.
shown	Specifies the length of the thumb as a fraction of the total length of the Scrollbar.

XtScrollbarThumb moves the visible thumb to position (0.0 – 1.0) and length (0.0 – 1.0). Either the top or shown arguments can be specified as –1.0, in which case the current value is left unchanged. Values greater than 1.0 are truncated to 1.0.

If called from the ThumbProc, XtScrollbarSetThumb has no effect.

To destroy a Scrollbar widget instance, use XtDestroyWidget and specify the widget ID for the Scrollbar.

3.5. Viewport Widget

The Viewport widget consists of a frame window, one or two Scrollbars, and an inner window. The frame window is determined by the viewing size of the data that is to be displayed and the dimensions to which the Viewport is created. The inner window is the full size of the data that is to be displayed and is clipped by the frame window. The Viewport widget controls the scrolling of the data directly. No application callbacks are required for scrolling.

When the geometry of the frame window is equal in size to the inner window, or when the data does not require scrolling, the Viewport widget automatically removes the Scrollbar(s). The forceBars option causes the viewport widget to display the Scrollbar(s) permanently.

The class variable for the Viewport widget is viewportWidgetClass.

When creating a Viewport widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNallowHoriz	Boolean	False	Flag to allow horizontal scroll bars
XtNallowVert	Boolean	False	Flag to allow vertical scroll bars
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	1	Width of the border in pixels

Name	Type	Default	Description
XtNdestroyCallback	XtCallbackList	NULL	Callback for XtDestroyWidget
XtNforceBars	Boolean	False	Flag to force display of scroll bars
XtNheight	int	widget-dependent	Height of the widget
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNsensitive	Boolean	True	Whether widget should receive input
XtNtranslations	TranslationTable	none	event-to-action translations
XtNuseBottom	Boolean	False	Flag to indicate bottom/top bars
XtNuseRight	Boolean	False	Flag to indicate right/left bars
XtNwidth	int	widget-dependent	Width of the widget
XtNx	int	0	x coordinate within parent
XtNy	int	0	y coordinate within parent

The Viewport widget manages a single child widget. When the size of the child is larger than the size of the Viewport, the user may interactively move the child within the Viewport by repositioning the Scrollbars.

To create a Viewport widget instance, use XtCreateWidget and specify the class variable viewportWidgetClass.

To insert a child into a Viewport widget, use XtCreateWidget and specify the widget ID of the previously created Viewport as the parent.

To remove a child from a Viewport widget, use XtUnmanageChild or XtDestroyWidget and specify the widget ID of the child.

To delete the inner window, any children, and the frame window, use XtDestroyWidget and specify the widget ID of the Viewport widget.

3.6. Box Widget

The Box widget provides geometry management of arbitrary widgets in a box of a specified dimension. The children are rearranged when resizing events occurs either on the Box or when children are added or deleted.

The children are arranged on a background that has its own specified dimensions and colors.

The class variable for the Box widget is boxWidgetClass.

When creating a Box widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	1	Border width on button box
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for XtDestroyWidget
XtNhSpace	int	4	Pixel distance left and right of children
XtNheight	int	NULL	Viewing height of inner window
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNtranslations	TranslationTable	none	event-to-action translations
XtNvSpace	int	4	Pixel distance top and bottom of children
XtNwidth	int	NULL	Viewing width of inner window
XtNx	int	0	Widget location x coordinate
XtNy	int	0	Widget location y coordinate

The Box widget positions its children in rows with hSpace pixels to the left and right of each child and vSpace pixels between rows. If the Box width is not specified, the Box widget uses the width of the widest child. After positioning all children, the Box widget attempts to shrink its size to the minimum dimensions required for the layout.

To create a box widget instance, use XtCreateWidget and specify the class variable boxWidgetClass.

To add a child to the Box, use XtCreateWidget and specify the widget ID of the Box as the parent of the new widget.

To remove a child from the Box, use XtUnmanageChild or XtDestroyWidget and specify the widget ID of the child.

To destroy a Box widget instance, use XtDestroyWidget and specify the widget ID of the Box widget. All the children of this box are automatically destroyed at the same time.

3.7. VPaned Widget

The VPaned widget manages children in a vertically tiled fashion. A region, called a grip, appears on the border between each child. When the pointer pointer is positioned on a grip and pressed, an arrow is displayed that indicates the direction the border can be moved While keeping the pointer button down, the user can move the pointer in the direction of the arrow. This, in turn, changes the window borders, causing one widget to shrink and a neighbor widget to grow. The VPaned widget can change the height of all vertical paned windows, which are arranged from the top to the bottom of the parent window.

The class variable for the VPaned widget is vPanedWidgetClass.

When creating a VPaned widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	1	Border width (pixels)
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for XtDestroyWidget
XtNforeground	Pixel	Black	Pixel value for the foreground color
XtNheight	int	NULL	Height of vPane
XtNgripIndent	int	16	Offset of grip from margin (pixels)
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNrefigureMode	Boolean	On	Whether vPane should adjust children
XtNsensitive	Boolean	True	Whether widget receives input
XtNtranslations	TranslationTable	none	event-to-action translations
XtNwidth	int	NULL	Width of vPane
XtNx	int	NULL	x position of vPane
XtNy	int	NULL	y position of vPane

To create a VPaned widget instance, use XtCreateWidget and specify the class variable vPanedWidgetClass.

Once the parent frame is created, you then add panes to it. Any type of widget can be paned.

To add a child pane to a VPaned frame, use XtCreateWidget and specify the widget ID of the VPaned widget as the parent of each new child pane.

When creating a child pane, the following resources, by which the VPaned widget controls the placement of the child, can be specified in the argument list or retrieved from the resource database:

Name	Type	Default	Description
XtNallowResize	Boolean	False	True if child is allowed to make resize requests
XtNmax	int	65535	maximum height for pane
XtNmin	int	1	minimum height for pane
XtNskipAdjust	Boolean	False	True if VPaned widget should not automatically resize pane

To delete a pane from a vertical-paned window frame, use XtUnmanageWidget or XtDestroyWidget and specify the widget ID of the child pane.

To enable or disable a child's request for pane resizing, use XtPanedAllowResize.

```
void XtPanedAllowResize(w, allow_resize)
    Widget w;
    Boolean allow_resize;
```

w	Specifies the widget ID of the child widget pane.
allow_resize	Enables or disables a pane window for resizing requests.

If allow_resize is True, VPane allows geometry requests from the child to change the pane's height. If allow_resize is False, VPane ignores geometry requests from the child to change the pane's height. The default state is True before the VPane is realized and False after it is realized.

To change the minimum and maximum height settings for a pane, use XtPanedSetMinMax.

```
void XtPanedSetMinMax(w, min, max)
    Widget w;
    int min, max;
```

w	Specifies the widget ID of the child widget pane.
min	New minimum height of the child, expressed in pixels.
max	New maximum height of the child, expressed in pixels.

To enable or disable automatic recalculation of pane sizes and positions, use XtPanedSetRefigureMode.

```
void XtPanedSetRefigureMode(w, mode)
    Widget w;
    Boolean mode;
```

w	Specifies the widget ID of the VPaned widget.
mode	Enables or disables refiguration.

You should set the mode to FALSE if you add or remove multiple panes to/from the parent frame after it has been realized, unless you can arrange to manage all the panes at once using XtManageChildren. After all the panes are added, set the mode to TRUE. This avoids unnecessary geometry calculations and ''window dancing''.

To delete an entire VPaned widget and all associated data structures, use XtDestroyWidget and specify the widget ID of the VPaned widget. All the children of the VPaned widget automatically are destroyed at the same time.

3.8. Form Widget

The Form widget can contain an arbitrary number of children or subwidgets. The Form provides geometry management for the subwidgets. Any combination of children can be added to a Form. When the Form is resized, it computes new positions and sizes for its children. This computation is based upon information provided when a child is added to the Form.

The class variable for a Form widget is formWidgetClass.

When creating a Form widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	1	Width of border in pixels
XtNdefaultDistance	int	4	Default value for XtNhorizDistance and XtNvertDistance
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for XtDestroyWidget
XtNheight	int	10	Height of form
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNsensitive	Boolean	True	Whether widget receives input
XtNtranslations	TranslationTable	none	event-to-action translations
XtNwidth	int	10	Width of form
XtNx	int	NULL	x position of form
XtNy	int	NULL	y position of form

To create a Form widget instance, use XtCreateWidget and specify the class variable formWidgetClass.

To add a new child to a Form, use XtCreateWidget and specify the widget ID of the previously created Form as the parent of the child.

When creating children that are to be added to a Form, the following additional resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbottom	XtEdgeType	XtRubber	See text
XtNfromHoriz	Widget	NULL	See text
XtNfromVert	Widget	NULL	See text
XtNhorizDistance	int	XtdefaultDistance	See text
XtNleft	XtEdgeType	XtRubber	See text
XtNresizable	Boolean	FALSE	TRUE if allowed to resize
XtNright	XtEdgeType	XtRubber	See text
XtNtop	XtEdgeType	XtRubber	See text
XtNvertDistance	int	XtdefaultDistance	See text

When a widget is added to a Form, it can, to some extent, specify hints to the form where it should be positioned within the form.

The parameters XtNhorizDistance and XtNfromHoriz let the widget position itself a specified number of pixels horizontally away from another widget in the form. As an example, XtNhorizDistance could equal 10 and XtNfromHoriz could be the widget ID of another widget in the Form. The new widget will be placed 10 pixels to the right of the widget defined in XtNfromHoriz. If XtNfromHoriz equals NULL, then XtNhorizDistance is measured from the left edge of the Form.

Similarly, the parameters XtNvertDistance and XtNfromVert let the widget position itself a specified number of pixels vertically away from another widget in the Form. If XtNfromVert equals NULL, then XtNvertDistance is measured from the top of the Form.

The XtNtop, XtNbottom, XtNleft, and XtNright parameters tell the Form where to position the widget when the Form is resized. The XtEdgeType structure contains:

```
typedef enum {
        XtChainTop,
        XtChainBottom,
        XtChainLeft,
        XtChainRight,
        XtRubber,
} XtEdgeType;
```

The XtChainTop, XtChainBottom, XtChainLeft, and XtChainRight fields maintain a constant distance from an edge of the widget to the top, bottom, left, or right edges of a Form. The XtRubber field maintains a proportional distance from an edge of a widget to all edges of the Form.

To remove a child from a Form, use XtUnmanageChild or XtDestroyWidget and specify the widget ID of the child widget.

To destroy a Form widget instance, use XtDestroyWidget and specify the widget ID of the Form. All children of the Form automatically are destroyed at the same time.

3.9. Dialog Widget

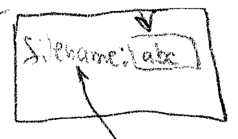
The Dialog widget implements a commonly used interaction semantic to prompt for auxiliary input from a user. For example, you can use a Dialog widget when an application requires a small piece of information, such as a file name, from the user. Essentially, a Dialog widget is a special case of the Form widget that provides a convenient way to create a "preconfigured form".

The typical Dialog widget contains three areas. The first line contains a description of the function of the Dialog widget, for example, the string "Filename:". The second line contains an area into which the user types input. The third line can contain buttons that let the user confirm or cancel the Dialog input.

The class variable for the Dialog box widget is dialogWidgetClass.

When creating a Dialog widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description
XtNbackground	Pixel	White	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	1	Width of border in pixels
XtNdestroyCallback	XtCallbackList	NULL	Callbacks for XtDestroyWidget
XtNheight	int	10	Height of dialog
XtNlabel	char*	label name	String to be displayed
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNmaximumLength	int	256	Maximum number of input characters
XtNsensitive	Boolean	True	Whether widget receives input
XtNtranslations	TranslationTable	none	event-to-action translations
XtNvalue	char*	NULL	Pointer to default string
XtNwidth	int	10	Width of dialog
XtNx	int	NULL	x position of dialog
XtNy	int	NULL	y position of dialog



The instance name of the label for the Dialog widget is "label", and the instance name of the Dialog value field is "value".

To create a Dialog widget instance, you can use XtCreateWidget and specify the class variable dialogWidgetClass, or you can use the XtDialogCreate convenience function.

```
Widget XtDialogCreate(parent, description, valueinit, args, num_args)
        Widget parent;
        char *description;
        char *valueinit;
        ArgList args;
        int num_args;
```

parent	Specifies the widget ID in which the Dialog box is to reside.
description	Specifies a pointer to the title for the Dialog box.
valueinit	Specifies a pointer to the initial value that is displayed in the Dialog box. Use NULL if you do not want a value field.
args	Specifies a variable-length argument list. The argument list is the same as that for the form widget.
num_args	Specifies the number of arguments in the argument list. When the argument list is NULL, the num_args must be zero. However, when the num_args is zero, the argument list does not have to be NULL.

ex: pre set up file name

XtDialogCreate is convenience function that calls XtCreateWidget and specifies the class dialogWidgetClass.

To add a child button to the Dialog box, use XtCreateWidget and specify widget ID of the previously created Dialog box as the parent of each child. When creating buttons, you do not have to specify form constraints. The Dialog box will automatically add the constraints.

To return the character string in the text field, use XtDialogGetValueString.

```
char *XtDialogGetValueString(w)
        Widget w;
```

w	Specifies the widget ID of the Dialog box.

To remove a child button from the Dialog box, use XtUnmanageChild or XtDestroyWidget and specify the widget ID of the child.

To destroy a Dialog widget instance, use XtDestroyWidget and specify the widget ID of the Dialog widget. All children of the Dialog automatically are destroyed at the same time.

3.10. Grip Widget

The Grip widget provides a small region in which pointer events are handled. The most common use for the grip is as an attachment point for visually repositioning an object, such as the pane border in a VPaned widget.

The class variable for the Grip widget is gripWidgetClass.

When creating a Grip widget instance, the following resources are retrieved from the argument list or from the resource database:

Name	Type	Default	Description

Name	Type	Default	Description
XtNbackground	Pixel	Black	Window background color
XtNbackgroundPixmap	Pixmap	none	Window background pixmap
XtNborderColor	Pixel	Black	Window border color
XtNborderPixmap	Pixmap	none	Window border pixmap
XtNborderWidth	int	0	Width of the border in pixels
XtNcallback	XtCallbackList	none	Action routine
XtNdestroyCallback	XtCallbackList	NULL	Callback for XtDestroyWidget
XtNheight	int	8	Height of the widget
XtNmappedWhenManaged	Boolean	True	Whether XtMapWidget is automatic
XtNsensitive	Boolean	True	Whether widget should receive input
XtNtranslations	TranslationTable	none	event-to-action translations
XtNwidth	int	8	Width of the widget
XtNx	int	0	x coordinate within parent
XtNy	int	0	y coordinate within parent

The Grip widget does not declare any default event translation bindings, but it does declare a single action routine named "GripAction" in its action table. Using this table, the client specifies an arbitrary event translation table giving parameters to the GripAction routine. The GripAction is declared in the client program.

The following is an example of a GripAction table:

```
<Btn1Down>:        GripAction(press)\n\
<Btn1Motion>:      GripAction(move)\n\
<Btn1Up>:          GripAction(release)
```

For a complete description of the format of action routines, see *X Toolkit Intrinsics – C Language X Interface.*

To create a Grip widget instance, use XtCreateWidget and specify the class variable gripWidgetClass.

To destroy a Command button widget instance, use XtDestroyWidget and specify the ID of the Grip widget.

X Window System, Version 11

Inter-Client Communication Conventions Manual

DRAFT 25th February 1988

David S. H. Rosenthal

Sun Microsystems
2550 Garcia Ave.
Mountain View CA 94043

ABSTRACT

It was an explicit design goal of the X Window System, Version 11[1] to specify mechanism, not policy. As a result, a client that converses with the server using the protocol defined by the *X Window System Protocol, Version 11* may operate "correctly" in isolation, but may not coexist properly with others sharing the same server. Conventions are proposed to allow clients to cooperate in the areas of selections, cut buffers, window management, session management, and resources.

"If you smile at me, you know that I will understand, 'cos that is something everybody everywhere does in the same language."
Jefferson Airplane, *Wooden Ships.*

1. Introduction

It was an explicit design goal of X11 to specify mechanism, not policy. As a result, a client that converses with the server using the protocol defined by the *X Window System Protocol, Version 11* may operate "correctly" in isolation, but may not coexist properly with others sharing the same server.

Being a good citizen in the X11 world involves adhering to conventions governing inter-client communications in a number of areas:

- The selection mechanism.
- The cut-buffers.
- The window manager.
- The session manager.

1. The X Window System is a Trademark of the Massachusetts Institute of Technology.
2. Permission to use, copy, modify and distribute this document for any purpose and without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies, and that the name of Sun Microsystems, Inc. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Sun Microsystems, Inc. makes no representations about the suitability of the software described herein for any purpose. It is provided "as is" without express or implied warranty.

- The resource database.
- The manipulation of shared resources.

In the following Sections we propose suitable conventions for each area, in so far as it is possible to do so *without* enforcing a particular user interface.

1.1. Status of the Selection Conventions

Apart from the obvious gaps in the specification of the target and type atom lists, the selection conventions described above have caused little controversy. This may reflect the fact that the selection mechanism is, as yet, little used.

It will obviously take some time for these conventions to obtain formal endorsement, and for the toolkits and clients to be changed to conform.

2. Peer-to-Peer Communication via Selections

The primary mechanism X11 defines for clients that want to exchange information, for example by cutting and pasting between windows, are *selections*. There can be an arbitrary number of selections, each named by an atom, and they are global to the server. The choice of an atom to be used is discussed later. Each selection is owned by a client, and is attached to a window.

Selections communicate between an *owner* and a *requestor*. The owner has the data representing the value of its selection, and the requestor receives it. A requestor wishing to obtain the value of a selection provides:

- the name of the selection
- the name of a property
- a window
- an atom representing the datatype required

If the selection is currently owned, the owner receives an event, and is expected to:

- convert the contents of the selection to the requested datatype
- place this data in the named property on the named window
- send the requestor an event to let it know the property is available.

Clients are strongly encouraged to use this mechanism. In particular, displaying text in a permanent window without providing the ability to select it and convert it into a string is definitely anti-social.

Note that, in the X11 environment, *all* data transferred between an owner and a requestor must normally go via the server. An X11 client cannot assume that another client can open the same files, or even communicate directly. The other client may be talking to the server via a completely different networking mechanism (for example, one client might be DECnet, and the other TCP/IP). Thus, passing indirect references to data such as file names, hostnames & port numbers, and so on is permitted only if both clients specifically agree.

2.1. Acquiring Selection Ownership

A client wishing to acquire ownership of a particular selection should call SetSelectionOwner:

```
SetSelectionOwner
      selection:  ATOM
      owner:            WINDOW or None
      time:       TIMESTAMP or CurrentTime
```

The client should set "selection" to the Atom representing the selection, set "owner" to some Window that it created and set "time" to some time between the current last-change time of the selection concerned and the current server time. This time value will normally be obtained from the timestamp of the event triggering the acquisition of the selection. Clients should *not* set the time value to CurrentTime, since if they do so they have no way of finding when they gained ownership of the selection. Clients must use a window they created, since SendEvent will be used with an empty mask to reply.

> Convention: *Clients attempting to acquire a selection must set the time value of the SetSelectionOwner request to the timestamp of the event triggering the acquisition attempt, not to CurrentTime. A zero-length append to a property is a way to obtain a time-stamp for this purpose, the timestamp is in the corresponding PropertyNotify event.*

Note that if the time in the SetSelectionOwner request is in the future relative to the server's current time, or if it is in the past relative to the last time the selection concerned changed hands, the SetSelectionOwner request appears to the client to succeed, but ownership is *not* actually transferred.

> Convention: *Clients are normally expected to provide some visible confirmation of selection. To make this feedback reliable, a client must perform the sequence:*
>
> <XXX - Xlib stuff>

```
XSetSelectionOwner(dpy, seln, own, time);
if (XGetSelectionOwner(dpy, seln) != own) {
  /* We didn't get the selection */
```

If the SetSelectionOwner request succeeds (not merely appears to succeed), the client issuing it is recorded by the server as being the owner of the selection for the time period starting at "time". Since clients cannot name other clients directly, they use the "owner" window to refer to the owning client in the replies to GetSelectionOwner, and in SelectionRequest and SelectionClear events, and possibly as a place to put properties describing the selection in question.

To discover the owner of a particular selection, a client should invoke:

```
GetSelectionOwner
      selection:   ATOM
=>
      owner:              WINDOW or None
```

> Problem: *There is no way for anyone to find out the last-change time of a selection. At the next protocol revision, GetSelectionOwner should be changed to return the last-change time as well as the owner.*

2.2. Responsibilities of the Selection Owner

When a requestor wants the value of a selection, the owner receives a SelectionRequest event:

```
SelectionRequest
      owner:              WINDOW
      selection:   ATOM
      target:             ATOM
      property:    ATOM or None
      requestor:   WINDOW
      time:        TIMESTAMP or CurrentTime
```

The owner and the selection fields will be the values specified in the SetSelectionOwner request. The owner should compare the timestamp with the period it has owned the selection and, if the time is outside, refuse the SelectionRequest by sending the requestor window a SelectionNotify event with the "property" set to None, using SendEvent with an empty event-mask.

More advanced selection owners are free to maintain a history of the value of the selection, and to respond to requests for the value of the selection during periods they owned it before the current one. Otherwise, the owner should use the "target" field to decide the form to convert the selection into, and if the selection cannot be converted into that form, refuse the SelectionRequest similarly.

If the "property" field is not None, the owner should place the data resulting from converting the selection into the specified property on the requestor window, setting the property's type to some appropriate value (which need not be the same as "target"). If the "property" field is None, the owner should choose a suitable property name and place the data as that property on the requestor window, setting the type as before.

> Convention: *All properties used to reply to SelectionRequest events must be placed on the requestor window.*

In either case, if the data comprising the selection cannot be stored on the requestor window (for example, because the server cannot provide sufficient memory), the owner must refuse the SelectionRequest as above. See the Section on "Large Data Transfers" below.

If the property is successfully stored, the owner should acknowledge the successful conversion by sending the requestor window a SelectionNotify event, using SendEvent with an empty mask:

```
SelectionNotify
        requestor:  WINDOW
        selection:  ATOM
        target:     ATOM
        property:   ATOM or None
        time:       TIMESTAMP or CurrentTime
```

The "selection", "target" and "property" fields should be set to the values received in the SelectionRequest event (Setting the "property" field to None indicates that the conversion requested could not be made).

> Convention: *The "selection", "target", "time" and "property" fields should be set to the values received in the SelectionRequest event.*

The data stored in the property must eventually be deleted. A convention is needed to assign the responsibility for doing so.

> Convention: *Selection requestors are responsible for deleting properties whose names they receive in SelectionNotify events (see Section 2.4). Owners are responsible for deleting all other properties involved in communicating selections.*

A selection owner will often need confirmation that the data comprising the selection has actually been transferred (for example, if the operation has side-effects on the owner's internal data-structures these should not take place until the data has been successfully received). They should express interest in PropertyNotify events for the "requestor" window and wait until the property in the SelectionNotify event has been deleted before assuming that the selection data has been transferred.

When some other client acquires a selection, the previous owner receives a SelectionClear event:

```
SelectionClear
        owner:      WINDOW
        selection:  ATOM
        time:       TIMESTAMP
```

The "timestamp" field is the time at which the ownership changed hands, and the "owner" field is the window the new owner specified in its SetSelectionOwner request.

If an owner loses ownership while it has a transfer in progress, that is to say before it receives notification that the requestor has received all the data, it must continue to service the on-going transfer until it is complete.

2.3. Giving Up Selection Ownership

2.3.1. Voluntarily

To relinquish ownership of a selection voluntarily, a client should execute a SetSelectionOwner request for that selection atom, with owner specified as None, and time specified as CurrentTime.

Alternatively, the client may destroy the window used as the "owner" value of the SetSelectionOwner request, or it may terminate. In both cases the ownership of the selection involved will revert to None.

2.3.2. Forcibly

If a client gives up ownership of a selection, or if some other client executes a SetSelectionOwner for it, the client will receive a SelectionClear event:

SelectionClear

owner:	WINDOW
selection:	ATOM
time:	TIMESTAMP

The timestamp is the time the selection changed hands. The owner argument is the window that was specified by the current owner in its SetSelectionOwner request.

2.4. Requesting a Selection

A client wishing to obtain the value of a selection in a particular form issues a ConvertSelection request:

ConvertSelection

selection:	ATOM
target:	ATOM
property:	ATOM or None
requestor:	WINDOW
time:	TIMESTAMP or CurrentTime

The selection field specifies the particular selection involved, and the target specifies the form the information is required in. The choice of suitable atoms to use is discussed below. The requestor field should be set to a window the requestor created; the owner will use it to place the reply property on. The time field should be set to the timestamp on the event triggering the request for the selection value, clients should *not* use CurrentTime for this field.

> Convention: *Clients should not use CurrentTime for the time field of ConvertSelection requests. They should use the timestamp of the event that caused the request to be made.*

The property field should be set to the name of a property that the owner can use to report the value of the selection, or to None (in which case the owner will chose a property name). Note that the requestor of a selection needs to know neither the owner of the selection, nor the window it is attached to.

> Convention: *Requestors should, wherever practicable, use None for the property field of ConvertSelection requests. By allowing the owner to choose the name of the reply property in this way, the need to coin new atoms can be reduced.*

The result of the ConvertSelection request is that a SelectionNotify event will be received:

SelectionNotify

requestor:	WINDOW
selection:	ATOM
target:	ATOM
property:	ATOM or None
time:	TIMESTAMP or CurrentTime

The "requestor", "selection", "time" and "target" fields will be the same as those on the Convert-Selection request.

If the "property" field is None, the conversion has been refused. This can mean that there is no owner for the selection, that the owner does not support the conversion implied by "target", or that the server did not have sufficient space to accomodate the data.

If the "property" field is not None, then that property will exist on the "requestor" window. The value of the selection can be retrieved from this property by using the GetProperty request:

```
GetProperty
        window:          WINDOW
        property:   ATOM
        type:       ATOM or AnyPropertyType
        long-offset: CARD32
        long-length: CARD32
        delete:          BOOL
=>
        type:       ATOM or None
        format:          {0, 8, 16, 32}
        bytes-after: CARD32
        value:      LISTofINT8 or LISTofINT16 or LISTofINT32
```

When using GetProperty to retrieve the value of a selection, the ''property'' field should be set to the corresponding value in the SelectionNotify event. The ''type'' field should be set to AnyPropertyType, because the requestor has no way of knowing beforehand what type the selection owner will use. Several GetProperty requests may be needed to retrieve all the data in the selection; each should set the ''long-offset'' field to the amount of data received so far, and the ''size'' field to some reasonable buffer size (see the Section on ''Large Data Transfers''). If the returned value of ''bytes-after'' is zero the whole property has been transferred.

Once all the data in the selection has been retrieved, which may require getting the values of several properties (see the Section on ''Selection Properties''), the property in the SelectionNotify should be deleted by invoking GetProperty with the ''delete'' field set True. As discussed above, the owner has no way of knowing when the data has been transferred to the requestor unless the property is removed.

> Convention: *The requestor must delete the property named in the SelectionNotify once all the data has been retrieved. They should invoke either DeleteProperty, or GetWindowProperty(delete==TRUE) after they have sucessfully retrieved all data comprising the selection. (See the Section on ''Large Data Transfers'' below.)*

2.5. Large Data Transfers

Selections can get large, and this poses two problems:

- Transferring large amounts of data to the server is expensive, and it would be beneficial to be able to reuse the data once it has been sent to answer further ConvertSelection requests.
- All servers will have limits on the amount of data that can be stored in properties. Exceeding this limit will result in a BadAlloc error on the ChangeProperty request that the selection owner uses to store the data.

To deal with the first problem, we define the following structure:

```
typedef struct {
        Window           window;
        Atom        property;
        unsigned long    start;
        unsigned long    length;
} XPropertyPart;
```

and establish the following conventions:

> Convention: *Selection owners should store the data describing a largish selection (where ''largish'' is defined flexibly, but is definitely less than the maximum-request-length in the connection handshake) in a property on the owner window, not on the requestor window. They should reply to the SelectionRequest with a property of type INDIRECT on the requestor window whose content is an array of XPropertyPart structures describing the parts of some other properties containing the selection. Normally, there will be a single entry in the array. The properties so named are the owner's responsibility, requestors should not delete them.*

Since the actual selection data remains in properties on the selection owner window, it can be used to reply to futher SelectionRequest events. However, owners must be careful not to modify the data in these properties between replying to the SelectionRequest and receiving the corresponding PropertyNotify indicating that the requestor has the data.

The problem of limited server resources is addressed by the following conventions:

> Convention: *Selection owners should transfer the data describing an immense selection (where "immense" is defined flexibly, but is larger than large-ish) using the INCREMENTAL property mechanism (see below).*
>
> Convention: *Any client using SetSelectionOwner to acquire selection ownership should arrange to process BadAlloc errors. For clients using Xlib, this involves using XSetErrorHandler() to override the default handler.*
>
> Convention: *A selection owner must confirm that no BadAlloc error occurred while storing the properties for a selection before replying with a confirming SelectionNotify event.*
>
> Convention: *When storing large amounts (relative to max-request-size) of data, clients should use a sequence of ChangeProperty(mode==Append) requests for reasonable quantities of data. This is to avoid locking-up servers and to limit the waste of data transfer caused by a BadAlloc error.*
>
> Convention: *If a BadAlloc error occurs during storing the selection data, all properties stored for this selection should be deleted, and the ConvertSelection request refused by replying with a SelectionNotify event with "property" set to None.*
>
> Convention: *In order to avoid locking-up servers for inordinate lengths of time, requestors retrieving large quantities of data from a property should perform a series of GetProperty requests, each asking for a reasonable amount of data.*

Single-threaded servers should be changed to avoid locking-up during large data transfers.

2.6. Usage of Selection Atoms

It is important to observe that defining a new atom consumes resources in the server, and they are not released until the server re-initializes. Thus, it must be a goal to reduce the need for newly minted atoms.

2.6.1. Selection Atoms

There can be an arbitrary number of selections, each named by an atom. To conform with the inter-client conventions, however, clients need deal with only these three selections:

- PRIMARY
- SECONDARY
- CLIPBOARD

Other selections may be used freely for private communciation among related groups of clients.

> Problem: *How does a client find out which selection atoms are valid?*

2.6.1.1. The PRIMARY Selection

The selection named by the atom PRIMARY is used for all commands which take only a single argument. It is the principal means of communication between clients which use the selection mechanism.

2.6.1.2. The SECONDARY Selection

The selection named by the atom SECONDARY is used:

- As the second argument to commands taking two arguments, for example "exchange primary and secondary selections."
- As a means of obtaining data when there is a primary selection, and the user does not wish to disturb it.

2.6.1.3. The CLIPBOARD Selection

The selection named by the atom CLIPBOARD is used to hold deleted data. Clients deleting data should:

- Assert ownership of the CLIPBOARD.
- Be prepared to respond to a request for the contents of the CLIPBOARD in the normal way, returning the deleted data. The request will be generated by the clipboard client described below.

Clients wishing to restore deleted data should request the contents of the CLIPBOARD selection in the usual way.

Except while a client is actually deleting data, the owner of the CLIPBOARD selection will be a single, special client implemented for the purpose. It should:

- Assert ownership of the CLIPBOARD selection.
- If it loses the selection (which will be because someone has some newly deleted data):
  - Obtain the contents of the selection from the new owner.
  - Re-assert ownership of the CLIPBOARD selection
- Respond to requests for the CLIPBOARD contents in the normal way.

2.6.2. Target Atoms

The atom that a requestor supplies as the "target" of a ConvertSelection request determines the form of the data supplied. The set of such atoms is extensible, but a generally accepted base set of target atoms is

Table 1 – Target Atoms and their Meanings

Atom	Meaning
TARGETS	list of valid target atoms
INDIRECT	look in the ConvertSelection property
STRING	Un-interpreted ISO Latin 1 text
PIXMAP	Pixmap ID
ODIF	ISO Office Document Interchange Format
POSTSCRIPT	PostScript* program
INTERPRESS	InterPress† program
OWNER_OS	operating system of owner
FILE_NAME	full pathname of a file
HOST_NAME	see WM_CLIENT_MACHINE
IP_ADDRESS	DARPA net address/port for owner
DECNET_ADDRESS	DECnet address for owning host
CHARACTER_POSITION	start and end of selection in bytes
LINE_NUMBER	start and end line numbers
COLUMN_NUMBER	
LENGTH	number of bytes in selection
USER	name of user running owner
PROCEDURE	name of selected procedure
MODULE	name of selected module
PROCESS	process ID of owner
TASK	task ID of owner
CLASS	class of owner - see WM_CLASS
NAME	name of owner - see WM_NAME
XXX	need to generate more & agree

needed. As a starting point for this, Table 1 contains those that have been suggested so far.

* PostScript is a Trademark of Adobe Systems Inc. † InterPress is a Trademark of Xerox.

Selection owners are required to support the following targets:

- TARGETS. They should return a list of the targets they are prepared to convert their selection into.
- INDIRECT. The INDIRECT target atom is valid only when a property is specified on the Convert-Selection request. If the property field in the SelectionRequest event is None and the target is INDIRECT, it should be refused.

  When a selection owner receives a SelectionRequest(target=INDIRECT) request, the contents of the property named in the request will be a list of atom pairs, the first atom naming a target, and the second naming a property (or None). The effect should be as if the owner had received a sequence of SelectionRequest events, one for each atom pair, except that;

  - the owner should reply with a SelectionNotify only when all the requested conversions have been performed.
  - the owner should replace any property atoms it received as None with the properties used to store the converted data.
  - the owner should replace any property atoms for targets it failed to convert with None.

All other targets are optional.

2.7. Usage of Selection Properties

The names of the properties used in selection data transfer are chosen by:

- The requestor, if the ''property'' field is not None.
- The selection owner, if the ''property'' field is None, and also in the case of all indirect properties.

The type of the properties involved is always chosen by the selection owner, and they can involve some types with special semantics assigned by convention. These special types are reviewed in the following Sections.

Clients receiving properties of these types should ensure that they have retreived all data they need from the other objects so named *before* they delete the property named in the reply.

2.7.1. STRING Properties

Clients receiving properties of type STRING can assume that for the purposes of displaying them the encoding is ISO Latin 1.

Character strings requiring other encodings are transmitted as properties with other types. Table 2 is a partial list of such types.

Table 2 – Property Types and Encodings

Type Atom	Meaning
STRING	ISO Latin 1
XXX	need more entries - who will generate?

2.7.2. INDIRECT Properties

Clients may receive properties of type INDIRECT. The contents of these properties will be arrays of window/atom pairs. The data in the selection consists of the data in each of the properties so named in turn, starting from the start of the array.

All selection requestors must be prepared to receive properties of type INDIRECT.

It is an error if the types of the second and subsequent properties in the list differ from the type of the first.

The properties named in the INDIRECT property are the owner's responsibility, requestors should not delete them.

2.7.3. INCREMENTAL Properties

Clients may receive properties of type INCREMENTAL. The contents of the property will be a window/atom pair, which the client and the selection owner will use to communicate using the following method.

The selection owner:

- appends the data in suitable-size chunks to the property. The size should be less than the maximum-request-size in the connection handshake. Between each append, the owner should wait for a PropertyNotify (state==Deleted) event showing that the requestor has read the data. The reason for doing this is to limit the consumption of space in the server.
- when the entire data has been transferred to the server, waits until a PropertyNotify (state==Deleted) showing that the data has been read by the requestor, and then writes zero-length data to the property.

The selection requestor:

- waits for the SelectionNotify event.
- loops:
  - retrieving data using GetProperty with "delete" True
  - waiting for a PropertyNotify with state==NewValue
- until a zero-length property is obtained
- deletes the type INCREMENTAL property

3. Peer-to-Peer Communication via Cut-Buffers

This Section is missing; Carnegie-Mellon has agreed to supply some information for it.

4. Client to Window-Manager Communciation

To permit window managers to perform their role of mediating the competing demands for resources such as screen space, the clients being managed must adhere to certain conventions, and must expect the window managers to do likewise. These conventions are covered here from the client's point of view, and again from the window manager's point of view in the *Window and Session Manager Conventions Manual.*

In general, these conventions are somewhat complex, and will undoubtedly change through time as new window management paradigms are developed. There is thus a strong bias towards defining only those conventions that are essential, and which apply generally to all window management paradigms. Clients designed to run with a particular window manager can easily define private protocols to add to these conventions, but must be aware that their users may decide to run some other window manager no matter how much the designers of the private protocol are convinced that they have seen the "one true light" of user interfaces.

It is a principle of these conventions that a general client should neither know nor care which window manager is running, or indeed if one is running at all. Each window manager will implement a particular window management policy; the choice of an appropriate window management policy for the user's circumstances is not one for an individual client to make but will be made by the user or the user's system administrator. This does not exclude the possibility of writing clients that use a private protocol to restrict themselves to operating only under a specific window manager, it merely ensures that no claim of general utility is made for such programs.

For example, the claim is often made "the client I'm writing is important, and it needs to be on top." Well, maybe it is important when it is being run for real, and it should then be run under the control of a window manager that recognizes "important" windows through some private protocol and ensures that they are on top. However, imagine that the "important" client is being debugged. Then, ensuring that it is always on top is no longer the appropriate window management policy, and it should be run under a window manager that allows other windows (e.g., the debugger) to appear on top.

For clients which need to scan the resource database for information about the window manager's resources, the res_class of the window manager is "Wm", and the res_name is the name of the window manager (e.g., "uwm"). Note that this differs from the normal convention, for two reasons:

- Clients should be able to discover the resources the window manager is using without being forced to know which window manager it is.
- At most one (top-level) window manager should be running, even if the server is driving several screens. The code for "wm" shows how a window manager can deal with multiple screens.

4.1. Client's Actions

In general, the object of the X11 design is that clients should as far as possible do exactly what they would do in the absence of a window manager, except for:

- Hinting to the window manager about the resources they would like to obtain.
- Cooperating with the window manager by accepting the resources they are allocated, even if they are not those requested.
- Being prepared for resource allocations to change at any time.

4.1.1. Creating a Top-Level Window

A client would normally expect to create its top-level windows as children of one or more of the root windows, using some boilerplate like:

```
win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
                        xsh.x, xsh.y, xsh.width, xsh.height,
                        bw, bd, bg);
```

or, if a particular one of the roots was required, like:

```
win = XCreateSimpleWindow(dpy, RootWindow(dpy, screen),
                        xsh.x, xsh.y, xsh.width, xsh.height,
                        bw, bd, bg);
```

Ideally, it should be possible to override the choice of a root window and allow clients (including window managers) to treat a non-root window as a pseudo-root. This would allow, for example, testing of window managers and the use of application specific window managers to control the sub-windows owned by the members of a related suite of clients.

To support this, we define the following extension to the semantics of XOpenDisplay() on UNIX-based† systems (similar extensions are required for other operating systems).

On UNIX-based systems, the display name or DISPLAY environment variable is a string that has the format:

hostname:*number*.*screen*.*prop*

If the prop component is present, it should be interpreted as the name of a property on the root window of each screen as returned in the connection handshake. If that property exists, and has the type SCREEN, its contents will be a PseudoScreen structure containing information that, from the client's point of view, should replace the information in the connection handshake describing this screen.

† UNIX is a trademark of Bell Laboratories.

```
typedef struct {
        Window root;            /* Root window id. */
        long width, height;     /* width and height of screen */
        long mwidth, mheight;   /* width and height of  in millimeters */
        VisualID root_visual;   /* root visual */
        Colormap cmap;          /* default colormap */
        unsigned long white_pixel;
        unsigned long black_pixel;      /* White and Black pixel values */
        long max_maps, min_maps;    /* max and min colormaps */
        long backing_store;     /* Never, WhenMapped, Always */
        Bool save_unders;
        long root_input_mask;   /* initial root input mask */
        Atom depths;            /* list of allowable depths on the screen */
} PseudoScreen;
```

The "depths" field is either None, in which case the handshake depths and visuals are valid, or the name of a property of type DEPTHS on the same window, which contains an array of PseudoDepth structures. The number of elements in this array controls the "ndepths" field of the Display structure.

```
typedef struct {
        long depth;         /* this depth (Z) of the depth */
        long nvisuals;              /* number of Visual types at this depth */
        Atom visuals;       /* list of visuals possible at this depth */
} PseudoDepth;
```

The depth of the Visual whose ID is "root_visual" controls the "root_depth" field of the Display structure. The "visuals" field is either None, in which case the handshake Visuals for this depth are valid, or the name of a property of type VISUALS on the same window, which contains an array of PseudoVisual structures:

```
typedef struct {
        VisualID visualid;/* visual id of this visual */
        long class;          /* class of screen (monochrome, etc.) */
        unsigned long red_mask, green_mask, blue_mask;        /* mask values */
        long bits_per_rgb;/* log base 2 of distinct color values */
        long map_entries; /* colormap entries */
} PseudoVisual;
```

4.1.2. Client Properties

Once the client has one or more top-level windows, it must place properties on that window to inform the window manager of its desired behaviour. Some of these properties are mandatory, and some are optional. Properties written by the client will not be changed by the window manager.

4.1.2.1. WM_NAME

The WM_NAME property is an un-interpreted string that the client wishes displayed in association with the window (for example, in a window headline bar).

The encoding used for this string (and all other un-interpreted string properties) is implied by the type of the property. Type STRING implies ISO Latin 1 encoding; for other types see the Section on "Usage of Selection Properties."

Window managers are expected to make an effort to display this information; simply ignoring WM_NAME is not acceptable behaviour. Clients can assume that at least the first part of this string is visible to the user, and that if the information is not visible to the user it is because the user has taken an explicit decision to make it invisible.

On the other hand, there is no guarantee that the user can see the WM_NAME string even if the window manager supports window headlines. The user may have placed the headline off-screen, or have covered it by other windows. WM_NAME should not be used for application-critical information, nor to announce asynchronous changes of application state that require timely user response. The expected uses are:

- To permit the user to identify one of a number of instances of the same client
- To provide the user with non-critical state information

Note that even window managers that support headline bars will place some limit on the length of string that can be visible; brevity here will pay dividends.

4.1.2.2. WM_ICON_NAME

The WM_ICON_NAME property is an un-interpreted string that the client wishes displayed in association with the window when it is iconified (for example, in an icon label). In other respects, it is similar to WM_NAME. Fewer characters will normally be visible in WM_ICON_NAME than WM_NAME, for obvious geometric reasons.

4.1.2.3. WM_NORMAL_HINTS

The WM_NORMAL_HINTS property is a XSizeHints structure describing the desired window geometry.

```
typedef struct {
        long flags;
        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;
        int base_width, base_height;
} XSizeHints;
```

Problem: *In this and other property structure definitions, it is assumed that sizeof (int) and sizeof (long) are 4.*

The definitions for the flags field are:

```
#define USPosition          (1L << 0)       /* user specified x, y */
#define USSize              (1L << 1)       /* user specified width, height */
#define PPosition           (1L << 2)       /* program specified position */
#define PSize               (1L << 3)       /* program specified size */
#define PMinSize            (1L << 4)       /* program specified minimum size */
#define PMaxSize            (1L << 5)       /* program specified maximum size */
#define PResizeInc          (1L << 6)       /* program specified resize increments */
#define PAspect             (1L << 7)       /* program specified min and max aspect ratios */
#define PBaseSize           (1L << 8)       /* progran specified base size */
```

The x, y, width, and height elements describe a desired position and size for the window. The coordinate system for x and y is the (pseudo-) root window, irrespective of any reparenting that may have occurred. To indicate that this information was specified by the user, set the **USPosition** and **USSize** flags. To indicate that it was specified by the client without any user involvement, set **PPosition** and **PSize**. This allows a window manager to know that the user specifically asked where the window should be placed or how the window should be sized and that the window manager does not have to rely the program's opinion.

The min_width and min_height elements specify the minimum size that the window can be for the client to be useful. The max_width and max_height elements specify the maximum size. The base_width and base_height elements in conjunction with width_inc and height_inc define an arithmetic progression of preferred window widths and heights:

width = base_width + (i * width_inc)
height = base_height + (j * height_inc)

for non-negative integers i and j. Window managers are encouraged to use i and j instead of width and height in reporting window sizes to users. If a base size is not provided, the minimum size is to be used in its place, and vice versa.

The min_aspect and max_aspect elements are expressed as ratios of x and y, and they allow a client to specify the range of aspect ratios it prefers.

4.1.2.4. WM_HINTS

```
typedef struct {
        long flags;
        Bool input;
        int initial_state;
        Pixmap icon_pixmap;
        Window icon_window;
        int icon_x, icon_y;
        Pixmap icon_mask;
        XID window_group;
        unsigned int messages;
} XWMHints;
```

The definitions for the flags field are:

```
#define  InputHint                    (1L << 0)
#define  StateHint                    (1L << 1)
#define  IconPixmapHint               (1L << 2)
#define  IconWindowHint               (1L << 3)
#define  IconPositionHint             (1L << 4)
#define  IconMaskHint                 (1L << 5)
#define  WindowGroupHint              (1L << 6)
#define  MessageHint                  (1L << 7)
```

The input field is used to communicate to the window manager the input focus model used by the client There are four such models:

- No Input. The client never expects keyboard input.

  An example would be xload, or another output-only client.

- Passive Input. The client expects keyboard input but never explicitly sets the input focus.

  An example would be a simple client with no subwindows, which will accept input in PointerRoot mode, or when the window manager sets the input focus to its top-level window (in click-to-type mode).

- Locally Active Input. The client expects keyboard input, and explicitly sets the input focus, but only does so when one of its windows already has the focus.

  An example would be a client with sub-windows defining various data entry fields, that uses Next and Prev keys to move the input focus between the fields, once its top-level window has acquired the focus in PointerRoot mode, or when the window manager sets the input focus to its top-level window (in click-to-type mode).

- Globally Active Input. The client expects keyboard input, and explicitly sets the input focus even when it is in windows the client does not own.

An example would be a client with a scroll bar that wants to allow users to scroll the window without disturbing the input focus even if it is in some other window. It wants to acquire the input focus when the user clicks in the scrolled region, but not when the user clicks in the scroll bar itself. Thus, it wants to prevent the window manager setting the input focus to any of its windows.

Clients with the Globally Active and No Input models should set the "input" flag to **False**. Clients with the Passive and Locally Active should set the "input" flag to **True**. For more details, see the FocusMessage bit below.

The definitions for the initial_state flag are:

```
#define  DontCareState        0
#define  NormalState          1
#define  ClientIconState      2
#define  IconicState          3
#define  InactiveState        4
#define  IgnoreState          5
```

The value of the initial_state flag determines the state the client wishes to be in at the time the top-level window is mapped. In addition, clients may ask the window manager to switch between states by setting the initial_state flag (its name has become somewhat misleading). The states are:

- DontCareState. The client doesn't care what state it is in.
- NormalState. The client wants its top-level window to be visible.
- ClientIconState. The client wants its icon_window to be visible. If icon_windows are not available, it wants its top-level window visible.
- IconicState. The client wants to be iconic, whatever that means for this window manager. It can assume that at least one of its icon_window (if any), its icon_pixmap (if any), or its WM_ICON_NAME will be visible.
- InactiveState. The client wants to be inactive, whatever that means for this window manager. Inactive windows will normally be available from a pop-up menu, or some other means that doesn't involve permanently allocating screen real-estate.
- IgnoreState. The client wants the window manager to ignore this window, and in return agrees that attempts to map it (or its icon_window) will be ignored.

The icon_pixmap field may specify a pixmap to be used as an icon. This pixmap should be:

- One of the sizes specified in the WM_ICON_SIZES property on the (pseudo-) root.
- 1-bit deep. The window manager will select, through the defaults database, suitable background (for the 0 bits) and foreground (for the 1 bits) colors. These defaults can, of course, specify different colors for the icons of different clients.

The icon_mask specifies which pixels of the icon_pixmap should be used as the icon, allowing for non-rectangular icons.

The icon_window field is the ID of a window the client wants used as its icon. Not all window managers will support icon windows, but those that do will expect that:

- The icon window is a child of the (pseudo-) root.
- The icon window is one of the sizes specified in the WM_ICON_SIZES property on the (pseudo-) root.
- The icon window uses the root visual & default colormap for the screen in question.
- The client will neither map nor unmap the icon window. If the client does map the icon window, it will be treated as if it had set the initial_state flag to ClientIconState. This treatment is not guaranteed to result in the icon window actually being mapped.

Clients needing more capabilities from the icons than a simple two-color bitmap should use icon windows.

The window_group lets the client specify that this window belongs to a group of windows. An example is a single client manipulating multiple children of the root window. Window managers may provide facilities for manipulating the group as a whole.

> Convention: *The window_group field should be set to the ID of the group leader. The window group leader may be a window which exists only for that purpose, and may never be mapped. Its window_group field should contain its own ID.*
>
> Convention: *The properties of the window group leader are those for the group as a whole (for example, the icon to be shown when the entire group is iconified).*

The definitions for the ''messages'' flags are:

```
#define    ConfigureDenied (1L << 0)   /* WM_CONFIGURE_DENIED */
#define    WindowMoved     (1L << 1)   /* WM_WINDOW_MOVED */
#define    BangMessage     (1L << 2)   /* BANG! */
#define    FocusMessage    (1l << 3)   /* WM_TAKE_FOCUS */
```

The meanings of the bits are:

- The ConfigureDenied bit enables notification that the window manager has decided not to change the size or position of the top-level window in response to a ConfigureWindow request.
- The WindowMoved bit enables notification that the window manager has moved a top-level window without resizing it.
- The BangMessage bit enables notification that the session manager feels that termination is likely.
- The FocusMessage bit announces that the client sets the input focus to its windows explicitly, and wants notification of when the window manager thinks it should do so.

If the ''messages'' field is not defined, it defaults to zeros.

The four input models and the corresponding values of the ''input'' and FocusMessage bits are shown in Table 3.

Table 3 – Input Models

Input Model	input field	FocusMessage
No Input	False	False
Passive	True	False
Locally Active	True	True
Globally Active	False	True

4.1.2.5. WM_CLASS

The WM_CLASS property is a string containing two null-separated elements, ''res_class'' and ''res_name'', that are meant to be used by clients both as a means of permanent identification, and as the handles by which which both the client and the window manager obtain resources related to the window. ''res_class'' is meant to identify the client (e.g., emacs), while ''res_name'' is meant to more specifically identify the particular instance. Resources should be obtained using ''res_name'', or if not found under ''res_name'', then using ''res_class''. The fields should be set using the following rules (non-Unix systems will differ):

- Set WM_CLASS.res_class to the name of the client (for example, ''emacs'').
- Set WM_CLASS.res_name from the first of the following that applies:
  - an optional command line argument (i.e., -rn name)
  - a specific environment variable (i.e., RESOURCE_NAME)
  - the trailing component of argv[0]

The WM_CLASS property is write-once and must be present when the window is mapped; window managers will ignore changes to it while the window is mapped. If the window is unmapped, and then re-mapped, window managers will normally re-read WM_CLASS. However, there should be no need for a client to change its class.

4.1.2.6. WM_TRANSIENT_FOR

The WM_TRANSIENT_FOR property is the ID of another top-level window. The implication is that this window is a pop-up on behalf of the named window, and window managers may decide not to decorate transient windows, or treat them differently in other ways. Dialogue boxes are an example of windows that should have WM_TRANSIENT_FOR set.

It is important not to confuse WM_TRANSIENT_FOR with override-redirect. WM_TRANSIENT_FOR should be used in those cases where the pointer is not grabbed while the window is grabbed; in other words, when other windows are allowed to be active while the transient is up. If other windows must be frozen, use override-redirect and grab the pointer while the window is mapped.

4.1.3. Window Manager Properties

The properties described above are those which the client is responsible for maintaining on its top-level windows. This Section describes the properties that the window manager places on clients top-level windows.

4.1.3.1. WM_STATE

The WM_STATE property is composed of two fields:

```
struct {
        int state;
        Window icon;
};
```

The state field can take on some of the same values as the initial_state field of the WM_HINTS property. In particular, it can be:

```
#define  NormalState            1
#define  ClientIconState        2
#define  IconicState            3
#define  InactiveState          4
```

The icon field should contain the window ID of the window which the window manager uses as the icon window for the window on which this property is set, if any, otherwise None. Note that this window may not be the same as the icon window which the client may specify.

Clients should be prepared for this property not being set for the standard reasons (no window manager, badly behaved window manager, broken window manager). The state field describes the window manager's idea of the state the window is in, which may not match the client's idea as expressed in the initial_state field of the WM_HINTS property (for example, if the user has asked the window manager to iconify the window). If it is NormalState, the window manager believes the client should be animating its window; if it is IconicState that it should animate its icon window. Note that in either state clients should be prepared to handle exposure events from either window.

4.1.4. Mapping and Unmapping the Window

Once the top-level window has been provided with suitable properties, the client is free to map it and unmap it as required. Mapping the window should be interpreted as a request to place this window under the control of the window manager, and unmapping it as a request to remove it from the control of the window manager. Typically, the window manager will intercept these calls and "do the right thing."

Note that mapping or unmapping the window is a heavyweight operation, and that it is *not* the way to change state from open to iconic, and vice versa. An unmapped window is not under the control of the window manager, and its icon will also be unmapped.

A client receiving UnmapNotify on its (top-level or icon) window should regard that as notification from the window manager that animating the window is no longer necessary. They should cease computing new states of the window, and stop sending output to it.

A client can also select for VisibilityChange on their (top-level or icon) windows. They will the receive a VisibilityNotify(state=FullyObscured) event when the window concerned becomes completely obscured even though mapped (and thus perhaps a waste of time to update), and a VisibilityNotify(state!=FullyObscured) when it becomes even partly viewable.

Clients should neither map nor unmap their icon windows.

4.1.5. Configuring the Window

There are two possible ways in which a client could resize or re-position its top-level windows:

<XXX - Xlib stuff>

- Use XConfigure().
- Use XSetNormalHints() to change the size and/or position fields in the WM_NORMAL_HINTS property.

Clients wishing to re-size or re-position their top-level windows should do *both* of them, as follows:

<XXX - Xlib stuff>

- Do an XConfigure() with the desired size and location.
- Do an XSetNormalHints() with the *same* size and location.

The order is important. The coordinate system in which the location is expressed is that of the (pseudo-) root, irrespective of any re-parenting that may have occurred (in this way clients need not be aware that they have been reparented).

Clients must be aware that there is no guarantee that the window manager will allocate them the requested size or location, and must be prepared to deal with *any* size and location. If the window manager decides to respond to a ConfigureRequest by:

- Not changing the size or location of the window at all. A client which has requested notification by setting the ConfigureDenied bit in WM_HINTS will receive a ClientMessage whose type field is the atom WM_CONFIGURE_DENIED, and which carries no other data.
- Moving the window without resizing it. A client which has requested notification by setting the WindowMoved bit in WM_HINTS will receive a ClientMessage whose type field is the atom WM_WINDOW_MOVED, and whose data field contains the new (pseudo-) root X and Y. They will not normally receive a ConfigureNotify event describing this change, since the window manager will have re-parented their window.
- Moving and resizing the window, a client which has selected for StructureNotify will receive a ConfigureNotify event. Note that the coordinates in this event are relative to the parent, which may not be the root in the window that has been reparented.

4.1.6. Input Focus

Clients can, as described above, deal with the input focus in four ways:

- No Input
- Passive
- Locally Active
- Globally Active

Passive and Locally Active clients set the "input" field of WM_HINTS **True** to indicate that they require window manager assistance in acquiring the input focus. No Input and Globally Active clients set the "input" field **False** to prevent the window manager setting the input focus to its top-level window.

Clients using SetInputFocus must set the "time" field to the timestamp of the event that caused them to make the attempt. Note that this cannot be a FocusIn event, since they do not have timestamps, and that clients may acquire the focus without a corresponding EnterNotify. Clients must not use CurrentTime in the "time" field.

Clients using the Globally Active model can only use SetInputFocus to acquire the input focus when they do not already have it on receipt of one of the following events:

- ButtonPress
- ButtonRelease
- Passive-grabbed KeyPress
- Passive-grabbed KeyRelease

In general, clients should avoid using pasive-grabbed Key events for this purpose except when they are unavoidable (for example, a selection tool that establishes a passive grab on the keys that cut, copy, or paste).

The method by which the user commands the window manager to set the focus to a window is up to the window manager. For example, clients cannot determine whether they will see the click that transfers the focus.

Clients which set the FocusMessage bit in WM_HINTS may receive a ClientMessage from the window manager whose "type" field is the atom WM_TAKE_FOCUS and whose data field is a timestamp. If they expect input, they should respond with a SetInputFocus request with its "window" field set to the window of theirs that last had the input focus, or to their "default input window", and the "time" field set to the timestamp.

Clients will normally receive WM_TAKE_FOCUS when opening from an icon, or when the user has clicked outside their window in an area that indicates to the window manager that it should assign the focus (for example, clicking in the headline bar can be used to asign the focus).

Clients that set the input focus need to decide a value for the "revert-to" field of the SetInputFocus request. This determines the behaviour of the input focus if the window the focus has been set to becomes not viewable. It can be any of:

- Parent. In general, clients should use this value when assigning focus to one of their subwindows. Unmapping the subwindow will cause focus to revert to the parent, which is probably what you want.
- PointerRoot. Using this value with a ClickToType-style window manager leads to race conditions, since the window becoming unviewable may coincide with the window manager deciding to move the focus elsewhere.
- None. Using this value causes problems if the window manager reparents the window (most window managers will) and then crashes. The input focus will be None, and there will probably be no way to change it.

There isn't a lot of experience to base a convention on, but the tentative convention is:

> Convention: *Clients invoking SetInputFocus should set "revert-to" to Parent.*

A convention is also required for clients that want to give up the input focus.

> Convention: *When a client with the focus wants to give it up, it should set it to None, rather than to PointerRoot.*

4.1.7. Colormaps

Clients that use one colormap for each top-level window and its sub-windows should set its ID in the colormap field of the window's attributes, and depend on the window manager to install it at suitable times. They should not set a WM_COLORMAPS property on the top-level window.

If they want to change the colormap, they should change the window attribute, and the window manager will install the colormap for them. Window managers are responsible for ensuring that top-level window colormaps are installed at appropriate times.

Clients that have sub-windows with different colormap requirements from their top-level windows should install these (sub-window) colormaps themselves. The window manager remains responsible for installing the top-level colormap. They should set the WM_COLORMAPS property on the top-level window concerned to a list of the IDs of the colormaps they will be installing. This informs the window manager that they will do their own sub-window colormap installation, and allows the window manager to uninstall other maps at suitable times.

Clients, especially those installing their own colormaps, should be aware of the min-installed-maps and max-installed-maps fields of the connection startup information, and the effect that the minimum value has on the "required list":

> "At any time, there is a subset of the installed maps, viewed as an ordered list, called the "required list". The length of the required list is at most M, where M is the min-installed-maps specified for the screen in the connection setup. The required list is maintained as follows. When a colormap is an explicit argument to InstallColormap, it is added to the head of the list, and the list is truncated at the tail if necessary to keep the length of the list to at most M. When a colormap is an explicit argument to UninstallColormap and it is in the required list, it is removed from the list. A colormap is not added to the required list when it is installed implicitly by the server, and the server cannot implicitly uninstall a colormap that is in the required list."

In less precise words, the min-installed-maps most recently installed maps are guaranteed to be installed. This number will often be one; clients needing multiple colormaps should beware.

The WM_COLORMAPS property is merely a hint to the window manager, allowing it to uninstall suitable maps when preparing to install a top-level window's map. If it is inconvenient for a client to collect the complete set of colormaps it will install, the property can be incomplete (or even empty). The only result of an incomplete list is that window manager's attempts to manage the set of installed maps will in some cases be less than optimal.

4.1.8. Icons

A client can hint to the window manager about the desired appearance of its icon in several ways:

- Set a string in WM_ICON_NAME. All clients should do this, as it provides a fall-back for window managers whose ideas about icons differ widely from those of the client.
- Set a Pixmap into the "icon_pixmap" field of the WM_HINTS property, and possibly another into the "icon_mask" field. The window manager is expected to display the pixmap masked by the mask. The pixmap should be one of the sizes found in the WM_ICON_SIZE property on the root (or pseudo-root). If this property is not found, the window manager is unlikely to display icon pixmaps. Window managers will normally clip or tile pixmaps which do not match WM_ICON_SIZE.
- Set a window into the "icon_window" field of the WM_HINTS property. The window manager is expected to map that window whenever the client is in IconState or ClientIconState. If the icon window is not the window itself, the window manager will treat the window's WM_NORMAL_SIZE property as a hint of a suitable icon size. In general, the size of the icon window should be one of those specified in WM_ICON_SIZE on the (pseudo-) root, if it exists. Window managers are free to resize icon windows.

> Convention: *Clients may ask the window manager to change their state from normal to iconic and vice versa by setting the "initial_state" flag in the WM_HINTS property.*

Clients must not depend on being able to receive input events via their icon windows. Window managers will differ as to whether they support this.

4.1.9. Popup Windows

Clients wishing to pop-up a window can do one of three things:

- They can create and map another normal top-level window, which will get decorated and managed as normal by the window manager. See the discussion of window groups below.
- If the window will be visible for a relatively short time, and deserves a somewhat lighter treatment, they can set the WM_TRANSIENT_FOR property. They can expect less decoration, but can set all the normal window manager properties on the window. An example would be a dialog box.
- If the window will be visible for a very short time, and should not be decorated at all, the client can set override-redirect on the window. In general, this should be done only if the pointer is grabbed while

the window is mapped. The window manager will never interfere with these windows, which should be used with caution. An example of an appropriate use is a pop-up menu.

4.1.10. Window Groups

A client with multiple persistent top-level windows constitutes a window group, and its top-level windows should be linked together using the ''window_group'' field of the WM_HINTS structure.

One of the windows (the one the others point to) will be the group leader and will carry the group as opposed to the individual properties. Window managers may treat the group leader differently from other windows in the group. For example, group leaders may have the full set of decorations, and other group members a restricted set.

It is not necessary for the group leader ever to be mapped.

4.2. Client Responses to Window Manager Actions

The window manager performs a number of operations on client resources, primarily on their top-level windows. Clients must not try to fight this, but may elect to receive notification of the window manager's operations.

4.2.1. Move

If the window manager moves a top-level window without changing its size, the client can elect to receive notification by setting the WindowMoved bit in the WM_HINTS structure. Notification is via a ClientMessage event whose type field is WM_WINDOW_MOVED and whose data field contains the new (pseudo-) root X and Y.

Clients must not respond to being moved by attempting to move themselves to a better location.

4.2.2. Resize

The client can elect to receive notification of being resized by selecting StructureNotify on its top-level window(s). A ConfigureNotify event on a top-level window implies that the window's position on the root may have changed, even though its position in its parent is unchanged, because the window may have been re-parented. And note that the coordinates in the event will not, in this case, be meaningful.

The response of the client to being resized should be to accept the size it has been given, and to do its best with it. Clients must not respond to being resized by attempting to resize themselves to a better size. If the size is impossible to work with, clients are free to change the ''initial_state'' field in their WM_HINTS structure and ask to be iconified.

4.2.3. (De)Iconify

Clients can know their open/closed status by examining the WM_STATE property on their top-level window. The window manager will set this, using the same bit definitions as the ''initial_state'' field in WM_HINTS, to indicate the state it believes the window is currently in.

Clients needing to take action on changing state (other than painting the window on the opening Expose event) can select for PropertyNotify and wait for notification of changes to the WM_STATE property.

If the only reason for wanting to know whether the window is in open or iconic state is to stop updating it when it is iconic, the client should select for StructureNotify on it, and stop updating the display when it receives UnmapNotify. UnmapNotify does not imply that the window is in any particular state, rather it implies that the window manager believes it is no longer necessary for the client to update the window. For example, consider a window that is open on a pseudo-root that is unmapped.

4.2.4. Colormap Change

Clients that wish to be notified of their colormaps being installed or uninstalled should select for Colormap-Notify on their top-level windows. They will receive ColormapNotify events with the ''new'' field FALSE when the colormap for that window is installed or uninstalled. Window managers will only ever install the colormaps for top-level windows, but a side-effect of them doing so (or of other clients installing subwindow colormaps) may be that other maps are uninstalled.

Clients that need to explicitly install colormaps for sub-windows should do so only when their top-level window has its colormap installed.

> Problem: *There is a race condition here; the InstallColormap request doesn't take a timestamp, and it may be executed after the top-level colormap has been uninstalled. The next protocol revision should provide the timestamp.*
>
> Convention: *Clients that need to install their own colormaps, and which expect input, should install them only when they have the input focus and their top-level colormap is installed. Clients that need to install their own colormaps, and which never expect input, should install them only when their top-level window has its colormap installed.*

4.2.5. Input Focus

Clients can request notification that they have the input focus by selecting for FocusChange on their top-level windows; they will receive FocusIn and FocusOut events. Clients that need to set the input focus to one of their sub-windows should not do so unless they actually have the focus in (one of) their top-level windows. Clients should not warp the pointer in an attempt to transfer the focus, they should set the focus and leave the pointer alone.

Once a client has the focus in one of its windows, it may transfer it to another of its windows using:

```
SetInputFocus
    focus:       WINDOW or PointerRoot or None
    revert-to:   {Parent, PointerRoot, None}
    time:        TIMESTAMP or CurrentTime
```

> Convention: *Clients using SetInputFocus must set the "time" field to the timestamp of the event that caused them to make the attempt. Note that this cannot be a FocusIn event, since they do not have timestamps, and that clients may acquire the focus without a corresponding EnterNotify. Clients must not use CurrentTime in the "time" field.*

4.2.6. ClientMessages

Clients may receive the following messages from their window manager. All will be events of type ClientMessage, distinguished by their "type" fields being the following atoms:

- WM_CONFIGURE_DENIED. These messages carry no data. They are enabled by setting the ConfigureDenied bit of the "messages" field of the WM_HINTS structure.
- WM_WINDOW_MOVED. These messages carry the new X and Y values in root window coordinates. They are enabled by setting the WindowMoved bit of the "messages" field of the WM_HINTS structure.
- WM_TAKE_FOCUS. These messages carry a timestamp. They are enabled by setting the FocusMessage bit of the WM_HINTS structure.

4.3. Status of the Window Manager Conventions

A number of areas in these conventions are still causing considerable controversy. Examples are:

- Input Focus handling
- WM_WINDOW_MOVED
- The XOpenDisplay() extension
- WM_ICON_SIZES
- WM_STATE

It will obviously take some time for these conventions to obtain formal endorsement, and for the toolkits and clients to be changed to conform.

5. Client to Session Manager Communication

The role of the session manager is to manage a collection of clients. It should be capable of:

- Starting a collection of clients as a group.
- Remembering the state of a collection of clients so that they can be re-started in the same state.
- Stopping a collection of clients in a controlled way.

It may also provide a user interface to starting, stopping, and re-starting groups of clients.

5.1. Client Actions

Clients need to cooperate with the session manager by providing it with information it can use to restart them if it should become necessary, but need not take any other action to assist it.

5.1.1. Properties

The client communicates with the session manager using two properties on its top-level window. If the client has a group of top-level windows, these properties should be placed on the group leader window.

5.1.1.1. WM_COMMAND

The WM_COMMAND property is a string representing the command used to (re-)start the client. In UNIX systems, it will initially be set from *argv*. Clients should ensure, by re-setting this property, that it always reflects a command that will restart them in their current state.

5.1.1.2. WM_CLIENT_MACHINE

The WM_CLIENT_MACHINE property should be set to a string forming the name of the machine running the client, as seen from the machine running the server.

5.1.2. Termination

Since they communicate via unreliable network connections, X11 clients must be prepared for their connection to the server to be terminated at any time without warning. They cannot depend on getting notification that termination is imminent, nor on being able to use the server to negotiate with the user (for example, using dialog boxes for confirmation) about their fate.

Equally, clients may terminate at any time without notice to the session manager.

5.2. Client Responses to Session Manager Actions

In general, the only action that a client need take in response to the actions of a session manager is to prepare for termination.

5.2.1. Termination

Clients that wish to be warned of impending termination should set the BangMessage bit in the WM_HINTS structure. They will receive a ClientMessage whose type field is BANG! whenever the session manager believes that termination is likely.

Clients that do not set the BangMessage bit may be terminated by the session manager at any time without warning.

Clients receiving BANG! should place themselves in a state from which they can be restarted, and should update WM_COMMAND to be a command that will restart them in this state. The session manager will be waiting for a PropertyNotify on WM_COMMAND as a confirmation that the client has saved its state, so that WM_COMMAND should be updated (perhaps with a zero-length append) even if its contents are correct.

Once it has received this confirmation, the session manager will feel free to terminate the client if that is what the user asked for. Otherwise, if the user merely asked for the session to be put to sleep, the session manager will ensure that the client does not receive any mouse or keyboard events.

Clients should regard BANG! not as a command to terminate, but rather as a warning from the session manager that it believes termination is likely. There is no need to terminate the session immediately but clients should be cognizant that the session manager may terminate them.

Nevertheless, a client is always free to terminate without giving either the session or window managers notice (and in particular is free to terminate when it receives the BANG! message). When a client terminates itself, rather than being terminated by the session manager, it is viewed as having resigned from the session in question, and it will not be revived if the session is revived.

After receiving a BANG! and saving its state, the client should not change its state until it receives a mouse or keyboard event. Once it does so, it can assume that the danger is over.

5.3. Status of the Session Manager Conventions

The question of client termination is still somewhat controversial. It will obviously take some time for these conventions to obtain formal endorsement, and for the toolkits and clients to be changed to conform.

6. Manipulation of Shared Resources

X11 permits clients to manipulate a number of shared resources, among them the input focus, the pointer, and colormaps. Conventions are required so that clients do so in an orderly fashion.

6.1. The Input Focus

Convention: *Clients should set the input focus to one of their windows only when it is already in one of their windows, or when they receive a WM_TAKE_FOCUS message. They should set the ''input'' field of the WM_HINTS structure TRUE.*

Convention: *Clients should use the timestamp of the event that caused them to attempt to set the input focus as the ''time'' field on the SetInputFocus request, not CurrentTime.*

6.2. The Pointer

In general, clients should not warp the pointer. Window managers may do so, for example to maintain the invariant that the pointer is always in the window with the input focus. Other window managers may wish to preserve the illusion that the user is in sole control of the pointer.

Convention: *Clients should not warp the pointer.*

Convention: *Clients which insist on warping the pointer should do so only with the ''src-window'' field of the WarpPointer request set to one of their windows.*

6.3. Colormaps

Convention: *Clients should install their own colormaps only if they have sub-windows with colormaps that differ from their top-level window.*

Convention: *Clients should warn the window manager that they will be installing their own colormaps by placing a list of the colormaps they will use in the WM_COLORMAPS property of the top-level window concerned.*

Convention: *Clients that need to install their own colormaps, and which expect input, should install them only when they have the input focus and their top-level colormap is installed. Clients that need to install their own colormaps, and which never expect input, should install them only when their top-level window has its colormap installed.*

Clients with DirectColor type applications should consult Section 9.2 of the Xlib manual for conventions connected with sharing standard colormaps.

7. Resource Manager Conventions

This Section has yet to be generated.

8. Conclusion

Acknowledgements

The author's role in preparing this document was limited to asking questions and writing down the answers; the ideas are all due to others. Sources for the questions were the Protocol and Xlib documents, and the answers came from:

- For the Selection Section, Jerry Farrell, Phil Karlton, and Bob Scheifler.
- For the Window and Session Manager Sections, Todd Brunhoff, Ellis Cohen, Hania Gajewska, Mark Manasse, Bob Scheifler, Raplh Swick, and the participants in the Window Manager discussions at the January 1988 X Conference at MIT.