Network Working Group
Request for Comments: 1013

Robert W. Scheifler

June 1987

# X WINDOW SYSTEM PROTOCOL, VERSION 11
## Alpha Update
## April 1987



## Status of this Memo

This RFC is distributed to the Internet community for information only. It does not establish an Internet standard. The X window system has been widely reviewed and tested. The internet community is encouraged to experiment with it. Distribution of this memo is unlimited (see copyright notice on page 2).

Author: Robert W. Scheifler
Laboratory for Computer Science
545 Technology Square, Room 418
Cambridge, MA 02139

Contributors:
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)

Implementors of initial server who provided useful input:
Susan Angebranndt (Digital)
Raymond Drewry (Digital)
Todd Newman (Digital)

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)
Craig Taylor (Sun)
Jeffery Vroom (Stellar)

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. Knowledge of X Version 10 will certainly aid in understanding this document.

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.

This document does not attempt to define precise formats or bit encodings.

---

## SECTION 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.

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.

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".

Bitmap
: A "bitmap" is a pixmap of depth one.

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 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 regions
: 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".

Color cell
: An entry in a colormap is known as a "color cell". An entry contains three values specifying red, green and blue intensities. These values are always viewed as 16 bit unsigned numbers, with zero being minimum intensity. The values are scaled by the server to match the display hardware. The components of a cell are coincident with components of other cells in DirectColor and TrueColor colormaps.

Colormap
: A "colormap" consists of a set of color cells. A pixel value indexes the color map to produce intensities to be displayed. 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 consist 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 pinter is in that window.

Depth
: The "depth" of a window or pixmap is number of bits per pixel it has. The depth of a gcontext is the depth of the root of the gcontext.

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

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 via "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 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 reconfigur 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.

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.

Graphics context
: Various information for graphics output is stored in "GC"'s, such as foreground pixel, background pixel, line width, clipping region, etc.

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 nominally 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.

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.

InputOutput window
: The "normal" kind of opaque window, used for both input and output.

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.

Mapping
: A window is said to be "mapped" if a map call has been performed on it. Unmapped windows are never viewable or visible.

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

Obscures
: 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 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.

Occludes
: 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 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, where N is the depth (z axis) of the pixmap. A pixmap can also be thought of as a stack of N bitmaps.

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 or 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 via properties.

Property list
: The "property list" of a window is the list of properties that have been defined for the window.

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 replys are multipexed on the same connection. Most requests do not generate 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.

Root
: The "root" of a pixmap or gcontext 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.

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.

Server
: The "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".

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.

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.

Unviewable
: A window is "unviewable" if it is mapped but some ancestor is unmapped.

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.

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.

## SECTION 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 enerated. 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 is implicitly assigned a sequence number, starting with one,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.

## SECTION 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.

Events in Section 12 are described in the following format:

```
    EventName
            value1: type1
            ...
            valueN: typeN

            Description.
```

## SECTION 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.

### BITMASK and 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; 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 Types

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.

```
DEVICE: 32-bit id (<class,model,manufacturer,unit> 8 bits each)
WINDOW: 32-bit id
PIXMAP: 32-bit id
CURSOR: 32-bit id
FONT: 32-bit id
GCONTEXT: 32-bit id
COLORMAP: 32-bit id
DRAWABLE: WINDOW or PIXMAP
ATOM: 32-bit id (top 3 bits guaranteed to be zero)
VISUALID: 32-bit id (top 3 bits guaranteed to be zero)
VALUE: 32-bit quantity (used only in LISTofVALUE)
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}
KEYCODE: CARD8
BUTTON: CARD8
KEYMASK: {Shift, CapsLock, 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, NS, ECMA, Datakit, DECnet}
       address: LISTofCARD8]
```

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 two 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 two-byte characters. It is expected that most "large" fonts will be defined with two-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.

## SECTION 5. ERRORS

In general, when a request terminates with an error, the request has no side effects (i.e., 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 (or otherwise authorized) host.

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 or <FONT or GCONTEXT> argument does not name a defined FONT.

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

IDChoice
: The value chosen for a resource identifier is either 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 which 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.

Match
: 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.

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.

Property
: The requested property does not exist for the specified window.

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 can generate this error.

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, e.g.,<Window or PointerRoot or None>.

## SECTION 6. KEYBOARDS

Keycodes are always in the inclusive range [8,255].

For keyboards with both left-side and right-side modifier keys (e.g., Shift and Control), the mask bits in the protocol always define the OR of the keys. If electronically distinguishable, they can have separate up/down events generated, and clients that want to distinguish can track the individual states manually.

<As part of the core we need to define a universal association between keycaps and keycodes. A keycap is the graphical information imprinted on a keyboard key, e.g., "$ 4", "T", "+ =".>

## SECTION 7. POINTERS

Buttons are always numbered starting with one.

## SECTION 8. PREDEFINED ATOMS

Predefined atoms are not strictly necessary, and may not be useful in all environments, but will eliminate many InternAtom requests in most applications. The core protocol imposes no semantics on these names,

except as they are used in FONTPROP structures (see QueryFont). Note that upper/lower case matters.

| | | |
|---|---|---|
| BITMAP | ICON_SIZE | RGB_GREEN_MAP |
| COMMAND | ITALIC_ANGLE | RGB_RED_MAP |
| COPYRIGHT | MAX_SPACE | SECONDARY |
| CUT_BUFFER0 | MIN_SPACE | SIZE_HINTS |
| CUT_BUFFER1 | NAME | STRIKEOUT_ASCENT |
| CUT_BUFFER2 | NORMAL_HINTS | STRIKEOUT_DESCENT |
| CUT_BUFFER3 | NORM_SPACE | STRING |
| CUT_BUFFER4 | PIXMAP | SUBSCRIPT_X |
| CUT_BUFFER5 | POINT_SIZE | SUBSCRIPT_Y |
| CUT_BUFFER6 | PRIMARY | SUPERSCRIPT_X |
| CUT_BUFFER7 | QUAD_WIDTH | SUPERSCRIPT_Y |
| DEFAULT_CHAR | RECTANGLE | UNDERLINE_POSITION |
| END_SPACE | RESIZE_HINT | UNDERLINE_THICKNESS |
| FACE_NAME | RESOLUTION | WEIGHT |
| FAMILY_NAME | RGB_BEST_MAP | WINDOW |
| FONT_ASCENT | RGB_BLUE_MAP | WM_HINTS |
| FONT_DESCENT | RGB_COLOR_MAP | X_HEIGHT |
| ICON | RGB_DEFAULT_MAP | ZOOM_HINTS |
| ICON_NAME | | |

## SECTION 9. CONNECTION SETUP

For remote clients, the X protocol can be built on top of any reliable byte stream. For TCP connections, displays on a given host a numbered starting from 0, and the server for display N listens and accepts connections on port 6000+N.

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 a server that only implements the host-based mechanism, will simply ignore this information.

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-format-scanline-unit: {8, 16, 32}
bitmap-format-scanline-pad: {8, 16, 32}
bitmap-format-bit-order: {LeastSignificant, MostSignificant}
pixmap-formats: LISTofFORMAT
roots: LISTofSCREEN
keyboard: DEVICE
pointer: DEVICE
motion-buffer-size: CARD32
maximum-request-length: CARD16
```

where

```
  FORMAT: [depth: CARD8,
```

```
                bits-per-pixel: {4, 8, 16, 24, 32}
                scanline-pad: {8, 16, 32}]
        SCREEN: [root: WINDOW
                device: DEVICE
                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 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.

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-format-scanline-pad. The

pad bits are of arbitrary value. The scanline is quantized in multiples of bits as given by bitmap-format-scanline-unit. Within each unit, the leftmost bit in the bitmap is either the least or most significant bit in the unit, as given by bitmap-format-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.

For each pixmap depth supported by some screen, pixmap-formats lists the ZFormat used to represent images of 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, and may be larger than strictly required by the depth. When the bits-per-pixel is 4, the order of nibbles in the byte is the same as the image byte-order. Each scanline is padded to a multiple of bits as given by 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 via the GetPointerMotions 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; i.e., 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 (i.e., requests of length up to and including 16384 bytes will be accepted by all servers).

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.

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 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. 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, then the a 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 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.

The bits-per-rgb-value specifies the log base 2 of the approximate number of distinct color values (individually) of red, green, and blue. Actual RGB values are always passed in the protocol within a 16-bit spectrum.

The colormap-entries defines the number of available colormap entries in a newly created colormap. For DirectColor and TrueColor, this will usually be the size of an individual pixel subfield.

## SECTION 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

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. 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:

```
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-bit-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:

```
background-pixmap: None
border-pixmap: CopyFromParent
bit-gravity: Forget
win-gravity: NorthWest
backing-store: NotUseful
backing-bit-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
```

Only the following attributes are defined for InputOnly windows: win-gravity, event-mask, do-not-propagate-mask, and cursor. It is a Match error to specify any other attributes for InputOnly windows.

If background-pixmap is given, it overrides the default background-pixel. 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 specifed, 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 Match error); 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 and any background-pixmap given, 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 regions of the window are exposed and the server has not retained the contents, the server automatically tiles the regions with the window's background unless the window has a background of None, in which case the previous screen contents are simply left in place. 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-pixel. 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 faster than others. If CopyFromParent is given, the parent's border pixmap is copied (subsequent changes to the parent do not affect the child), but the window must have the same depth as the parent (else a Match error). If border-pixel is given, it overrides the default and any border-pixmap given, 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. Note that, even if the window is larger than its parent, the server should maintain complete contents, not just the region within the parent boundaries. If the server maintains contents, Exposure events will not be generated, but the server may stop maintaining contents at any time. A value of NotUseful advises the server that maintaining contents is unnecessary, although a server may still choose to maintain contents.

Backing-bit-planes indicates (with one bits) which bit planes of the window hold dynamic data that must be preserved in backing-stores. Backing-pixel specifies what value to use in planes not covered by backing-bit-planes. The server is free to only save the specified bit planes in the backing-store, and regenerate the remaining planes with the specified pixel value.

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.

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.

Override-redirect specifies whether map and configure request 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 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 parents’s colormap is copied (subsequent changes to the parent do not affect the child), but the window must have the same visual type as the parent (else a Match error) an the parent must not have a colormap of None (else a Match error).

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 display 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.

Changing the background does not cause the window contents to be changed. Setting the border, or changing the background such that 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 back-store of an obsecured window to WhenMapped or Always, or changing the backing-bit-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 ResizeRedirectr, and at most one client at a time can select for ButtonPress.

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

Changing the colormap of a window (i.e., defining a new map, not changing the contents of the existing map) generates a ColormapNorify event. Changing the colormap os a visible window may 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-bit-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

Returns current attributes of the window. 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, in order from the argument window downwards, with unspecified order among siblings at each level.

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

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

ChangeSaveSet
window: WINDOW
mode: {Insert, Delete}

Errors: Window, Match, Value

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). The use of the save-set is described in 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, or if the new parent is the

window itself or an inferior of the window, or if the 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, then 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

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

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

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

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. 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, then 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 width or height of the window is being changed, then a ResizeRequest event is generated, and the current width and height are used instead in the following.

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 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.

Changing the 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:

```
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 GravityNotify 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; 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.

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, else 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, else 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.

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

            Returns the root and (current) geometry of the drawable.
            Depth is the number of bits per pixel for the object.
            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

            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

            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 ASCII 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
```

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

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.

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.

DeleteProperty
window: WINDOW
property: ATOM

Errors: Window, Atom

Deletes the property from the specified window if the property exists. 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
format: {8, 16, 32}
bytes-after: CARD32
value: LISTofINT8 or LISTofINT16 or LISTofINT32

Errors: Window, Atom, Property, Match, Value

If the specified property does not exist for the specifed window, a Property error is generated. Otherwise, if type AnyPropertyType is specified, (part of) the property is returned regardless of its type; if a type is specified, (part of) the property is returned only if its type equals the specified type (else a Match error). The actual type and format of the property are returned.

Define 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 0), and its length in bytes is L. 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 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: INT8
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).

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

            Returns the atoms of properties currently defined on the
            window.

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

            Error: Atom, Window

            Changes the owner and last-change time of the specifed
            selection.  The 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 new owner is not the same as the current owner of the
            selection, and the current owner is a window, then the
            current owner is sent a SelectClear event.

            If the owner of a selection is a window, and the window 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 server.

            Selections are global to the server.

GetSelectionOwner
            selection: ATOM
        =>
            owner: WINDOW or None

            Errors: Atom

            Returns the current owner of the specified selection, if any.

ConvertSelection
            selection, target: ATOM
```

property: ATOM or None
requestor: WINDOW
time: TIMESTAMP or CurrentTime

Error: Atom, Window

If the specified selection is owned by a window, the server sends a SelectionRequest event to the 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, and otherwise destination is replaced with the focus window.

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 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.

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

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.

Event-mask is always augmented to include ButtonPress and ButtonRelease. 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.

Pointer-mode controls further processing of pointer events, and keyboard-mode controls further processing of keyboard events. If the mode is Asynchronous, event processing continues normally; if the device is currently frozen by this client, then processing of events for the device is resumed. If the mode is Synchronous, the device (as seen via the protocol) appears to freeze, and no further events for that device are generated by the server until the grabbing client issues a releasing AllowEvents request. Actual device changes are not lost while the device 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

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 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 specified button is pressed when the specified modifier keys are down (and no other buttons or modifier keys are down), and grab-window contains the pointer, and the confine-to window (if any) is viewable, and these constraints are not satisfied for any ancestor, then the pointer is actively grabbed as described in GrabPointer, the last-pointer-grab time is set to the time at which the button was pressed (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 all buttons are released (independent of the state of modifier keys).

A modifiers of AnyModifier is equivalent to issuing the request for all possible modifier combinations. A button of AnyButton is equivalent to issuing the request for all possible buttons.

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

UngrabButton

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

Errors: Window

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. A button of AnyButton is equivalent to issuing the request for all possible buttons. Has no effect on an active grab.

ChangeActivePointerGrab

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

Errors: Cursor

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. The event-mask is always augmented to include ButtonPress and ButtonRelease. 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

Actively grabs control of the keyboard. Further key events are reported only to the grabbing client. The 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.

Pointer-mode controls further processing of pointer events, and keyboard-mode controls further processing of keyboard events. If the mode is Asynchronous, event processing continues normally; if the device is currently frozen by this client, then processing of events for the device is resumed. If the mode is Synchronous, the device (as seen via the protocol) appears to freeze, and no further events for that device are generated by the server until the grabbing client issues a releasing AllowEvents request. Actual device changes are not lost while the device 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

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 AnyNonModifier
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 specified key (which can itself be a modifier key) is pressed when the specified modifier keys are down (and no other modifier keys are down), and the KeyPress event would be generated in grab-window or one of its inferiors, and these constraints are not satisfied for any ancestor, then the keyboard is actively grabbed as described in GrabKeyboard, the last-keyboard-grab time is transmitted in set to the time at which the key was pressed (as 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 specified key has been released (independent of the state of the modifier keys).

A modifiers of AnyModifier is equivalent to issuing the request for all possible modifier combinations. A key of AnyNonModifier is equivalent to issuing the request for

all possible non-modifier key codes.

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 AnyNonModifier, the request fails completely (no grabs are established) if there is a conflicting grab for any combination.

UngrabKey

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

Errors: Window

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. A key of AnyNonModifier is equivalent to issuing the request for all possible non-modifier key codes. Has no effect on an active grab.

AllowEvents

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

Errors: Value

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 from a previous 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 pointer is frozen twice by the client on behalf of two separate grabs, AsyncPointer "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 notfrozen as the result of an event.

AsyncPointer, SyncPointer, and Replay Pointer 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). 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 due to both grabs. In this case, the freeze must be released on behalf of both grabs before events can again be processed.

GrabServer

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

UngrabServer

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 currently 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 state of the modifier keys and the buttons are also returned.

GetMotionEvents

```
            start, stop: TIMESTAMP or CurrentTime
            window: WINDOW
       =>
            events: LISTofTIMECOORD

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

            Error: Window
```

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.

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 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
src-x, src-y: INT16
src-width, src-height: CARD16
dst-x, dst-y: INT16

Errors: Window

Moves the pointer to [dst-x, dst-y] relative to dst-window's origin. If src-window is None, the move is independent of the current pointer position, but if a window is specified, the move only takes place if the pointer is currently contained in a visible portion of the specified rectangle of the 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.

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

          Errors: Window, Value

          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,
          therevert-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.

          If the focus window 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}

          Returns the current focus state.

QueryKeymap
      =>
          keys: LISTofCARD8
```

Returns a bit vector for 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.

OpenFont

```
fid: FONT
name: STRING8
```

Errors: IDChoice, Name, Alloc

Loads the specified font, if necessary, and associates identifier fid with it. The font can be used as a source for any drawable. The font name should use the ASCII encoding, and upper/lower case does not matter.

CloseFont

```
font: FONT
```

Errors: Font

Deletes the association between the resource id and the font. The font itself will be freed when no other resource references it.

QueryFont

```
font: FONT or GCONTEXT
=>
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: INT32 or CARD32]
        CHARINFO: [left-side-bearing: INT16
                   right-side-bearing: INT16
                   character-width: INT16
                   ascent: INT16
                   descent: INT16
                   attributes: CARD16]
```

Errors: Font

Returns logical information about a font.

The draw-direction is essentially just a hint, indicating 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 elementb 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 two-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 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 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, i.e., 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. Whether a property value is signed or unsigned must be derived from a prior 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 that upper/lower case matters).

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

SUBSCRIPT_X: INT32
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 + SubscriptX, y + SubscriptY].

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 + UnderlinePosition).

UNDERLINE_THICKNESS: CARD32
Thickness of the underline, in pixels.

STRIKEOUT_ASCENT: INT32
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 - StrikeoutAscent), and the height of the box is (StrikeoutAscent + StrikeoutDescent).

ITALIC_ANGLE: INT32
The angle 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.
        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.  There are 72.27 points to the
            inch.
        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,i.e., 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 undefined by the core protocol.

QueryTextExtents
        font: FONT or GCONTEXT
        items: 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
```

Returns the logical extents of the specified string of characters in the specified font. Draw-direction, font-ascent, and font-descent are as described in QueryFont. 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. 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. 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 two-byte matrix indexing, the server will interpret each CHAR2B as a 16-bit number that has been transmitted most significant byte first (i.e., 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
```

Returns a list of length at most max-names, of names of fonts matching the pattern. The pattern should use the ASCII 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 lower case.

```
ListFontsWithInfo
           pattern: STRING8
           max-names: CARD16
       =>
           fonts: LISTofFONTDATA

           where
                   FONTDATA: [name: STRING8
                              info: FONTINFO]
                   FONTINFO: <same type definition as in QueryFont>
```

Like ListFonts, but 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).

```
SetFontPath
           path: LISTofSTRING8

           Errors: Value
```

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
```

Returns the current search path for fonts.

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

           Errors: IDChoice, Drawable, Value, Alloc
```

Creates a pixmap, and assigns the identifier pid to it. Width and height must be non-zero. Depth must be one of the depths supported by root of the specified drawable. 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

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

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.

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

```
alu-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
```

```
  subwindow-mode: {ClipByChildren, IncludeInferiors}
  graphics-exposures: BOOL
  clip-x-origin: INT16
  clip-y-origin: INT16
  clip-mask: PIXMAP or None
  dash-offset: CARD16
  dash-list: 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 alu-functions are:

| | |
|---|---|
| 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 |

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)]

where 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 (for example, Bresenham). 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, but this property is not guaranteed for thin lines. Also note that "jags" in adjacent wide lines will always line up properly, but this property is not guaranteed for thin lines. 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 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 segments 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 segments defined by the even dashes are filled differently than the segments defined by the odd dashes (see fill-style)

OnOffDash: only the segments defined by the even dashes are drawn, and cap-style applies to each

individual segment (except NotLast is treated as Butt for internal caps)

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, 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

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 stipple operations, 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 (PolyText8, PolyText16, PolyFillRectangle, FillPoly, PolyFillArc), for line requests (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 dash-list 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. The meaning of dash-offset and dash-list 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. 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. 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). 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 drawing through subwindow boundaries. The use of IncludeInferiors on a window of one depth with mapped inferiors of differing depth is not illegal, but the semantics isundefined by the core protocol.

The fill-rule defines what pixels are inside (i.e., 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 counterclockwise directed path segments. For both 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:

```
    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
    full-rule: EvenOdd
    arc-mode: PieSlice
    tile: pixmap of unspecified size filled with forground
          pixell (i.e., client specified pixel if any,
          else 0)
    stipple: pixmap of unspecified size filled with ones
    tile-stipple-x-origin: 0
    tile-stipple-y-origin: 0
    font: <implementation dependent>
    subwindow-mode: ClipByChildren
    graphics-exposures: True
    clip-x-origin: 0
    clip-y-origin: 0
    clip-mask: None
    dash-offset: 0
    dash-list: 4 (i.e., the list [4, 4])
```

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

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 the dash-offset or dash-list 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

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
dash-list: LISTofCARD8

Errors: GContext, Value, Alloc

Sets the dash-offset and dash-list in gc for dashed line styles. The initial and alternating elements of the dash-list are the "even" dashes, the others are the "odd" dashes. All of the elements must be non-zero. The dash-offset defines the phase of the pattern, specifying how many pixels into the dash-list the pattern should actually begin in any single graphics request. Dashing is continuous through path segments combined with a join-style, but is reset to the dash-offset each time a cap-style is applied.

SetClipRectangles

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

Errors: GContext, Value, Alloc, Match

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.

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 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

Deletes the association between the resource id and the gcontext, and destroys the gcontext.

ClearToBackground

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 alu-function of Copy. If the window has background None, the contents of the window are not changed. In eithercase, if exposures is True, then one or more exposure events are generated for regions of the rectangle that are eithervisible 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

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, dst-x and dst-y are relative to dst-drawable's origin, each pair specifying the upper left corner of the rectangle. 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 by the server, or if regions outside the boundaries of the source drawable are specified, then the following occurs. If the dst-drawable is a window with a background of other than None, the corresponding regions of the destination are tiled (with plane-mask of ones and alu-function Copy) with that background. Regardless, if graphics-exposures in gc is True, GraphicsExposure events for the corresponding desitnation regions are generated.

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

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

CopyPlane

scr-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

Src-drawable must have the same root as dst-srawable (else a match error), but need not have the same depth.
Bit-plane must have exactly one bit set. Effectively, that plane of the src-drawable and the fore-ground/background pixels in gc are combined to form a pixmap of the same depth as dst-drawable, and the equivalent of a CopyArea is

performed, with all the same exposure semantics.

GC components: alu-function, plan-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

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.

GCcomponents: alu-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

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: alu-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,dash-list

PolySegment

drawable: DRAWABLE
gc: GCONTEXT
segments: LISTofSEGMENT

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

Errors: Drawable, GContext, Match

For each segment, 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: alu-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, dash-list

PolyRectangle

drawable: DRAWABLE
gc: GCONTEXT
rectangles: LISTofRECTANGLE

Errors: Drawable, GContext, Match

Draws the outlines of the specified rectangles, as if a five-point PolyLine were specified for each rectangle. 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: alu-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, dash-list

PolyArc

drawable: DRAWABLE
gc: GCONTEXT
arcs: LISTofARC

Errors: Drawable, GContext, Match

Draws 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-oclock position from the center, 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 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: alu-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, dash-list

FillPoly

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

Errors: Drawable, GContext, Match, Value

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: alu-function, plane-mask, fill-style, fill-rule, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask

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

PolyFillRectangle

drawable: DRAWABLE
gc: GCONTEXT
rectangles: LISTofRECTANGLE

Errors: Drawable, GContext, Match

Fills the specified rectangles. The x and y coordinates of each rectangle are relative to the drawable's origin, and define the upper left corner of the rectangle.

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

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

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

PolyFillArc

drawable: DRAWABLE
gc: GCONTEXT
arcs: LISTofARC

Errors: Drawable, GContext, Match

For each arc, fills the region closed 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: alu-function, plane-mask, fill-style, fill-rule, arc-mode, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask

GC mode-dependent components: foreground, 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}
bits: <bits>

Errors: Drawable, GContext, Match, Value, Alloc

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, 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. For Bitmap and XYPixmap format, left-pad must be less than bitmap-format-scanline-pad (as given in the server connection setup info). 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: alu-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: {XYFormat, ZFormat}

=>

depth: CARD8
visual: VISUALID or None
bits: <bits>

Errors: Drawable, Value, Match

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 XYFormat is specified, only the bit planes specified in plane-mask are transmitted. If ZFormat is specified, then bits in all planes not specified in plane-mask transmitted as zero. The returned depth specifies the number of bits per pixel of the image. 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 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 will include any visible portions of inferiors or overlapping windows contained in the rectangle, but if these windows are of different depth than the specified window, the contents returned for them are not defined by the core protocol.

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 with differing draw-directions is permitted. 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 (not added or subtracted based on the draw-direction of the current font). Each character image, as defined by the a 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: alu-function, plane-mask, fill-style, font, subwindow-mode, clip-x-origin, clip-y-origin, clip-mask

GC mode-dependent components: foreground, 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-x: INT8
                                string: STRING16]

            Errors: Drawable, GContext, Match, Font
```

Just like PolyText8, except two-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 (i.e., 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 to first fill a destination rectangle with the background pixel defined in gc, and then paint the text with the foreground pixel. The upper left corner of the filled rectangle is at

```
        [x + overall-left, y - font-ascent]
```

the width is

```
        overall-right - overall-left
```

and the height is

```
        font-ascent + font-descent
```

where overall-left, overall-right, font-ascent, and as font-descent are would be returned by a QueryTextExtents call using gc and string.

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

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: 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 two-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 (i.e., 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

           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, and cannot be of class TrueColor
           (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, but entries cannot be
           freed with FreeColors, and no relationships among entries
           is defined; the client must understand whether the colormap
           is GrayScale, PseudoColor, or DirectColor to know how to
           store into entries.

FreeColormap
           cmap: COLORMAP

           Errors: Colormap

           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 (via 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.

Has no effect on a default colormap for a screen.

CopyColormapAndFree
mid, src-cmap: COLORMAP

Errors: Colormap, Alloc

Creates a colormap for the same screen as src-cmap, and associates identifier mid with it. Moves all of the client's existing allocations from src-cmap to the new colormap, and frees those entries in src-cmap. Values in other entries in the new colormap are undefined.

InstallColormap
cmap: COLORMAP

Errors: Colormap

Makes this colormap an installed map for its screen. All windows associated with this colormap immediately display with true colors. As a side-effect, previously installed colormaps may be uninstalled, and other windows may display with false colors. Which colormaps get uninstalled is server dependent, except that it is guaranteed that the M-1 most recently client-installed colormaps will not be uninstalled, where M is the min-installed-maps specified for the screen in the connection setup.

If cmap is not already an installed map, a ColormapNotify event is generated on every window having cmap as an attribute. If a colormap is uninstalled as a result of the install, a ColormapNotify event is generated on every window having that colormap as an attribute.

Initially only the default colormap for a screen is installed.

UninstallColormap
cmap: COLORMAP

Errors: Colormap

If cmap is an installed map for its screen, one or more colormaps are installed in its place; the choice is server

dependent, pexcept that if the screen's default colormap is not installed and can be installed (without forcing other colormaps out), then the default colormap is used.

If cmap is an installed map, a ColormapNotify event is generated on every window having this colormap as an attribute. If a colormap is installed as a result of the uninstall, a ColormapNotify event is generated on every window having that colormap as an attribute.

```
ListInstalledColormaps
            window: WINDOW
        =>
            cmaps: LISTofCOLORMAP

            Errors: Window
```

Returns a list of the currently installed colormaps for the screen of the specified window.

```
AllocColor
            cmap: COLORMAP
            red, green, blue: CARD16
        =>
            pixel: CARD32
            red, green, blue: CARD16

            Errors: Colormap, Alloc
```

Allocates a read-only colormap entry corresponding to the closest RGB values provided by the hardware. Returns the pixel and the RGB values actually used.

```
AllocNamedColor
            cmap: COLORMAP
            name: STRING8
        =>
            pixel: CARD32
            exact-red, exact-green, exact-blue: CARD16
            screen-red, screen-green, screen-blue: CARD16

            Errors: Colormap, Name, Alloc
```

Looks up the named color with respect to the screen associated with the colormap, then does an AllocColor on cmap. The name should use the ASCII encoding, and upper/lower case does not matter. The exact RGB values specify the "true" values for the color, and the screen 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, the number of planes non-negative. 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 one bit, and for DirectColor each will have exactly three 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, the reds, greens, and blues non-negative. 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. 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 (i.e., 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
```

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

            Looks up the named color with respect to the screen
            associated with cmap, then does a StoreColors in cmap.  The
            name should use the ASCII encoding, and upper/lower case
            does not matter.

QueryColors
            cmap: COLORMAP
            pixels: LISTofCARD32
        =>
            colors: LISTofRGB

            where
                    RGB: [red, green, blue: CARD16]

            Errors: Colormap, Value

            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
            screen-red, screen-green, screen-blue: CARD16

            Errors: Colormap, Name

            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.
            The name should use the ASCII 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, Bitmap, Match, Value, Alloc

Creates a cursor and associates identifier cid with it. Foreground and background RGB values must be specified, even if the server only has a monochrome 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 can have any root. The mask pixmap defines the shape of the cursor; that is, the one bits in the mask define which source pixels will be displayed. If no mask is given, all pixels of the source are displayed. The mask, if present, must be the same size as source (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

Similar to CreateCursor, but the source and mask bitmaps are obtained from the specified font glyphs. The mask font and character are optional. The origin of the source glyph defines the hotspot, and the mask is positioned such that the origins are coincident. The source and mask need not have the same bounding box metrics. 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.

FreeCursor
cursor: CURSOR

```
            Errors: Cursor

            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

            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

            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

            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 ASCII encoding, and upper/lower case matters.

```
ListExtensions
       =>
           names: LISTofSTRING8
```

Returns a list of all extensions supported by the server.

```
SetKeyboardMapping
           map: LISTofCARD8
       =>
           status: {Success, Busy}

           Errors: Value
```

Sets the mapping of the keyboard. Elements of the list are indexed starting from one. The list must be of length 255. The index is a "core" keycode, and the element of the list defines the "effective" keycode.

A zero element disables a key, no elements can have values 1 through 7, and no two elements (with index larger than 7) can have the same non-zero value. If the keyboard does not really generate a given keycode, specifying a non-zero value for that core keycode has no effect.

Elements 6 and 7 of the map must always be zero. The first five elements are special: they specify the keycodes (if any) that correspond to the Mod1 through Mod5 modifiers. Setting one of these entries to zero disables use of that modifier bit. No two of the firstfive elements can have the same non-zero value.

A server can impose restrictions on how keyboards get remapped, e.g., if certain keys do not generate up transitions in hardware.

If any of the keys or modifiers to be altered are currently in the down state, the status reply is Busy and the mapping is not changed.

```
GetKeyboardMapping
       =>
           map: LISTofCARD8
```

Errors: Value

Returns the current mapping of the keyboard. Elements of the list are indexed starting from one. The length of the list is 255.

The nominal mapping for a keyboard is almost the identity mapping, except that map[i]=0 for keycodes that have no corresponding physical key, and the first five entries indicate the keycodes (if any) corresponding to the Mod1 through Mod5 modifier bits.

ChangeKeyboardControl
value-mask: BITMASK
value-list: LISTofVALUE

Errors: Match Value

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

```
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}
```

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.

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.

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.

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 are supported, numbered from one. 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

Errors: Match

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. 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: Match, Value

Rings the bell on the keyboard at the specified volume relative to the base volume for the keyboard, if possible. Percent, which can range from -100 to 100 inclusive, is added to the base volume, and the sum limited to the range 0 to 100

inclusive.

```
SetPointerMapping
            map: LISTofCARD8
       =>
            status: {Success, Busy}

            Errors: Value
```

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. 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.

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

```
GetPointerMapping
       =>
            map: LISTofCARD8

            Errors: Value
```

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: Match, Value
```

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 may 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

            Errors: Match

            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

            Timeout and interval are specified in minutes; 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 minutes, 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, randomly re-origined each interval
            minutes if the interval value is non-zero.  Otherwise, the
            state of the screen does 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.

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

            Returns the current screen-saver control values.

ForceScreenSaver
            mode: {Activate, Reset}

            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

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 in the initial authorization at connection setup.

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

ListHosts
=>
mode: {Enabled, Disabled}
hosts: LISTofHOST

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.

ChangeAccessControl
mode: {Enable, Disable}

Errors: Value, Access

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 in the initial authorization at connection setup.

ChangeCloseDownMode
mode: {Destroy, RetainPermanent, RetainTemporary}

Errors: Value

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, 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, allowing 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.

## SECTION 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 eleased. If the client has the server grabbed, and UngrabServer is performed. If close-down mode (see ChangeCloseDownMode) 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

created by the client, that 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 unmaped, a MapWindow request is performed on it. After save-set processing, all windows created by the client are destroyed. For each non-window resource created by the client, the appropriate Free request is performed. All colors and colormap entries allocated by the client are freed.

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, 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.

## SECTION 12. EVENTS

When a button is pressed with the pointer in some window W, and no active pointer grab is in progress, then the ancestors if 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:

event-window: the event window
event-mask: the client's selected 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 all buttons are released. UngrabPointer and ChangeActiveGrab can both be used to modify the active grab.

KeyPress
and
KeyRelease
and

ButtonPress
and
ButtonRelease
and
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

Generated when a key or button changes state, or the pointer moves. 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 the focus window. The window the event is reported with respect to is called the "event" window.

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 the source window. The state component gives the state of the buttons and modifier keys just before the event. The detailcomponent varies with the event type:

| | |
|---|---|
| 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 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
and
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 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 LeaveNotifyreceive LeaveNotify events. The pointer position reported in the event is always the "final" position, not the "initial" position of the pointer. 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 the event window is the focus window or an inferior of the focus window, then focus is True, and otherwisefocus 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.

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
LeaveNotify with detail NonlinearVirtual is generated on each window above A up to and including its root (in order)
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), 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, 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
and
FocusOut

event: WINDOW

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

Generated when the input focus changes. 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.

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
- FocusOut with detail NonlinearVirtual is generated on each window above A up to and including its root (in order)
- 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)

- 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 above A up to and including its root (in order)
- FocusIn with detail PointerRoot (or None) is generated on all root windows

When the focus moves from PointerRoot (or None) to window A:

- FocusOut with detail PointerRoot (or None) is generated on all root windows
- 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):

- FocusOut with detail PointerRoot (or None) is generated on all root windows
- FocusIn with detail None (or PointerRoot) is generated on all root windows

When a keyboard grab activates, 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, 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. Reported to clients selecting KeymapState on a window. Generated immediately after every EnterNotify and FocusIn.

Expose
window: WINDOW
x, y, width, height: CARD16
last-in-series: BOOL

Reported to clients selecting Exposure on the window. Possibly generated when a region of the window becomes viewable, but might only be generated when a region becomes visible. All of the regions exposed by a given "action" are guaranteed to be reported contiguously; if last-in-series is False then another exposure follows.

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

Expose events are never generated on InputOnly windows.

GraphicsExposure
drawable: DRAWABLE
x, y, width, height: CARD16
last-in-series: BOOL
major-opcode: CARD8
minor-opcode: CARD16

Reported to clients selecting graphics-exposures in a graphics context. 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 last-in-series is False then another exposure follows.

The x and y coordinates are relative to drawable's origin, and specify the upper left corner of a rectangule. 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

Reported to clients selecting graphics-exposures in a graphics context. 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 minor-opcode is always zero.

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

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.

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

Reported to clients selecting SubstructureNotify on the parent. Generated when the window is created. The arguments are as in the CreateWindow request.

DestroyNotify
event, window: WINDOW

Reported to clients selecting StructureNotify on the window, and to clients selecting SubstructureNotify on the parent. Generated when the window is destroyed. "Event" is the window on which the event was generated, and "window" is the window that is destroyed.

UnmapNotify
event, window: WINDOW
from-configure: BOOL

Reported to clients selecting StructureNotify on the window, and to clients selecting SubstructureNotify on the parent. Generated when the window changes state from mapped to unmapped. "Event" is the window on which the event was generated, and "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

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

MapRequest
parent, window: WINDOW

Reported to the client selecting SubstructureRedirect on the parent. 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

Reported to clients selecting SubstructureNotify on either the old or the new parent, and to clients selecting StructureNotify on the window. Generated when the window is reparented. "Event" is the window on which the event was generated, "window" is the window that has been re-rooted, and "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

Reported to clients selecting StructureNotify on the window, and to clients selecting SubstructureNotify on the parent. Generated when a ConfigureWindow request actually changes the state of the window. "Event" is the window on which the event was generated, and "window" is the window that is changed. 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

Reported to clients selecting SubstructureNotify on the parent, and to clients selecting StructureNotify on the window. Generated when a window is moved because of a change in size of the parent. "Event" is the window on which the event was generated, and "window" is the window that is moved.

ResizeRequest
window: WINDOW
width, height: CARD16

Reported to the client selecting ResizeRedirect on the window. 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
above-sibling: WINDOW or None

Reported to the client selecting SubstructureRedirect on the parent. Generated when a ConfigureWindow request is issued on

the window by some other client. The geometry is as derived from the request. The above-sibling is the sibling the window should be placed directly on top of; if None, then the window should be placed on the bottom.

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

Reported to clients selecting StructureNotify on the window, and to clients selecting SubstructureNotify on the parent. Generated when the window is actually restacked from a CirculateWindow request. "Event" is the window on which the event was generated, and "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}

Reported to the client selecting SubstructureRedirect on the parent. 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 place specifies what the new position in the stacking order should be.

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

Reported to clients selecting PropertyChange on the window. Generated when a property of the window is changed. The timestamp indicates the server time when the property was changed.

SelectionClear
owner: WINDOW
selection: ATOM
time: TIMESTAMP

Reported to the current owner of a selection. Generated on the window losing ownership when a new owner is being defined. The timestamp is the last-change time recorded for the selection.

SelectionRequest
owner: WINDOW

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

Reported to the owner of a selection. Generated when a client issues a ConvertSelection request. The arguments are as in the 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. 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 only generated by clients using SendEvent. 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 property None).

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

Reported to clients selecting ColormapChange on the window. Generated with value True for new when the colormap attribute of the window is changed. Generated with value False for new when the colormap of a window is installed or uninstalled. In either case, state indicates whether the colormap is currently installed.

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.

## SECTION 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 the transport layer has very little buffering, or unless 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 (i.e., 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 does not include the actual transmission of the reply and the events. In addition, the effect of any other "cause" (e.g., 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.