X Window System Protocol

MIT X Consortium Standard

X Version 11, Release 4

Robert W. Scheifler

Massachusetts Institute of Technology
Laboratory for Computer Science

Acknowledgments

The primary contributers to the X11 protocol are:

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

The implementors of initial server who provided useful input are:

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

The invited reviewers who provided useful input are:

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

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

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

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

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

1. Protocol Formats

Request Format

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

Reply Format

Every reply contains a 32-bit length field expressed in units of four 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.

2. Syntactic Conventions

The rest of this document uses the following syntactic conventions.

- The syntax {...} encloses a set of alternatives.
- The syntax [...] encloses a set of structure components.
- In general, TYPEs are in uppercase and **AlternativeValues** are capitalized.
- Requests in section 10 are described in the following format:

RequestName

arg1: type1
...
argN: typeN
=>
result1: type1

...
resultM: typeM

Errors: kind1, ..., kindK

Description.

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

- Events in section 12 are described in the following format:

EventName
value1: type1
...
valueN: typeN

Description.

3. 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), and in some cases, it may be implicit. It is fully specified in Appendix B. Except where explicitly noted, zero-length lists are legal.

BITMASK
LISTofVALUE

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

value-mask: BITMASK
value-list: LISTofVALUE

These are used to allow the client to specify a subset of a heterogeneous collection of optional arguments. The value-mask specifies which arguments are to be provided; each such argument is assigned a unique bit position. The representation of the BITMASK will typically contain more bits than there are defined arguments. The unused bits in the value-mask must be zero (or the server generates a **Value** error). The value-list contains one value for each bit set to 1 in the mask, from least-significant to most-significant bit in the mask. Each value is represented with four bytes, but the actual value occupies only the least-significant bytes as required. The values of the unused bytes do not matter.

OR

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

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

PIXMAP: 32-bit value (top three bits guaranteed to be zero)

CURSOR: 32-bit value (top three bits guaranteed to be zero)

FONT: 32-bit value (top three bits guaranteed to be zero)

GCONTEXT: 32-bit value (top three bits guaranteed to be zero)

COLORMAP: 32-bit value (top three bits guaranteed to be zero)

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

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

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

BOOL: {**True**, **False**}

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

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

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

KEYSYM: 32-bit value (top three bits guaranteed to be zero)
KEYCODE: CARD8
BUTTON: CARD8
KEYMASK: {**Shift**, **Lock**, **Control**, **Mod1**, **Mod2**, **Mod3**, **Mod4**, **Mod5**}
BUTMASK: {**Button1**, **Button2**, **Button3**, **Button4**, **Button5**}
KEYBUTMASK: KEYMASK or BUTMASK
STRING8: LISTofCARD8
STRING16: LISTofCHAR2B
CHAR2B: [byte1, byte2: CARD8]

POINT: [x, y: INT16]

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

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

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

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

The primary interpretation of large characters in a STRING16 is that they are composed of 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 2-byte characters. It is expected that most large fonts will be defined with 2-byte matrix indexing. For large fonts constructed with linear indexing, a CHAR2B can be interpreted as a 16-bit number by treating byte1 as the most-significant byte. This means that clients should always transmit such 16-bit character values most-significant byte first, as the server will never byte-swap CHAR2B quantities.

The length, format, and interpretation of a HOST address are specific to the family (see **ChangeHosts** request).

4. Errors

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

The following error codes result from various requests as follows:

Error	Description
Access	An attempt is made to grab a key/button combination already grabbed by another client. An attempt is made to free a colormap entry not allocated by the client. An attempt is made to store into a read-only or an unallocated colormap entry. An attempt is made to modify the access control list from other than the local host (or otherwise authorized client). An attempt is made to select an event type that only one client can select at a time when another client has already selected it.

Error	Description
Alloc	The server failed to allocate the requested resource. Note that the explicit listing of **Alloc** errors in request only covers allocation errors at a very coarse level and is not intended 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, but a server may generate an **Alloc** error on any request for this reason, and clients should be prepared to receive such errors and handle or discard them.
Atom	A value for an ATOM argument does not name a defined ATOM.
Colormap	A value for a COLORMAP argument does not name a defined COLORMAP.
Cursor	A value for a CURSOR argument does not name a defined CURSOR.
Drawable	A value for a DRAWABLE argument does not name a defined WINDOW or PIXMAP.
Font	A value for a FONT argument does not name a defined FONT. A value for a FONTABLE argument does not name a defined FONT or a defined GCONTEXT.
GContext	A value for a GCONTEXT argument does not name a defined GCONTEXT.
IDChoice	The value chosen for a resource identifier either is not included in the range assigned to the client or is already in use.
Implementation	The server does not implement some aspect of the request. A server that generates this error for a core request is deficient. As such, this error is not listed for any of the requests, but clients should be prepared to receive such errors and handle or discard them.
Length	The length of a request is shorter or longer than that required to minimally contain the arguments. The length of a request exceeds the maximum length accepted by the server.
Match	An **InputOnly** window is used as a DRAWABLE. In a graphics request, the GCONTEXT argument does not have the same root and depth as the destination DRAWABLE argument. Some argument (or pair of arguments) has the correct type and range, but it fails to match in some other way required by the request.
Name	A font or color of the specified name does not exist.
Pixmap	A value for a PIXMAP argument does not name a defined PIXMAP.
Request	The major or minor opcode does not specify a valid request.

Error	Description
Value	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives typically can generate this error (due to the encoding).
Window	A value for a WINDOW argument does not name a defined WINDOW.

Note

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

5. Keyboards

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

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

A list of KEYSYMs is associated with each KEYCODE. The list is intended to convey the set of symbols on the corresponding key. If the list (ignoring trailing NoSymbol entries) is a single KEYSYM ‘‘*K*’’, then the list is treated as if it were the list ‘‘*K* NoSymbol *K* NoSymbol’’. If the list (ignoring trailing NoSymbol entries) is a pair of KEYSYMs ‘‘*K1 K2*’’, then the list is treated as if it were the list ‘‘*K1 K2 K1 K2*’’. If the list (ignoring trailing NoSymbol entries) is a triple of KEYSYMs ‘‘*K1 K2 K3*’’, then the list is treated as if it were the list ‘‘*K1 K2 K3* NoSymbol’’. When an explicit ‘‘void’’ element is desired in the list, the value VoidSymbol can be used.

The first four elements of the list are split into two groups of KEYSYMs. Group 1 contains the first and second KEYSYMs, Group 2 contains third and fourth KEYSYMs. Within each group, if the second element of the group is NoSymbol, then the group should be treated as if the second element were the same as the first element, except when the first element is an alphabetic KEYSYM ‘‘*K*’’ for which both lowercase and uppercase forms are defined. In that case, the group should be treated as if the first element were the lowercase form of ‘‘*K*’’ and the second element were the uppercase form of ‘‘*K*’’.

The standard rules for obtaining a KEYSYM from a **KeyPress** event make use of only the Group 1 and Group 2 KEYSYMs; no interpretation of other KEYSYMs in the list is given here. Which group to use is determined by modifier state. Switching between groups is controlled by the KEYSYM named MODE SWITCH, by attaching that KEYSYM to some KEYCODE and attaching that KEYCODE to any one of the modifiers Mod1 through Mod5. This modifier is called the ‘‘group modifier’’. For any KEYCODE, Group 1 is used when the group modifier is off, and Group 2 is used when the group modifier is on.

Within a group, which KEYSYM to use is also determined by modifier state. The first KEYSYM is used when the Shift and Lock modifiers are off. The second KEYSYM is used when the Shift modifier is on, or when the Lock modifier is on and the second KEYSYM is uppercase alphabetic, or when the Lock modifier is on and is interpreted as ShiftLock. Otherwise, when the Lock modifier is on and is interpreted as CapsLock, the state of the Shift modifier is applied first to select a KEYSYM, but if that KEYSYM is lowercase alphabetic, then the corresponding uppercase KEYSYM is used instead.

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

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

6. Pointers

Buttons are always numbered starting with one.

7. Predefined Atoms

Predefined atoms are not strictly necessary and may not be useful in all environments, but they will eliminate many **InternAtom** requests in most applications. Note that they are predefined only in the sense of having numeric values, not in the sense of having required semantics. The core protocol imposes no semantics on these names, but semantics are specified in other X Consortium standards, such as the ‘‘Inter-Client Communication Conventions Manual’’ and the ‘‘X Logical Font Description Conventions’’.

The following names have predefined atom values. Note that uppercase and lowercase matter.

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

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

8. Connection Setup

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

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

Following the byte-order byte, the client sends the following information 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.
>
> 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 meantime, a server that implements a different protocol than the client expects or that only implements the host-based mechanism may simply ignore this information. If both name and data strings are empty, this is to be interpreted as ‘‘no explicit authorization.’’

The client receives the following information 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 four 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 11, and the minor version will be 0. 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.

The client receives the following additional data if authorization fails:

> reason: STRING8

The client receives the following additional data if authorization is accepted:

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

where:

FORMAT:	[depth: CARD8, bits-per-pixel: {1, 4, 8, 16, 24, 32} scanline-pad: {8, 16, 32}]
SCREEN:	[root: WINDOW width-in-pixels, height-in-pixels: CARD16 width-in-millimeters, height-in-millimeters: CARD16 allowed-depths: LISTofDEPTH root-depth: CARD8 root-visual: VISUALID default-colormap: COLORMAP white-pixel, black-pixel: CARD32 min-installed-maps, max-installed-maps: CARD16 backing-stores: {**Never**, **WhenMapped**, **Always**} save-unders: BOOL current-input-masks: SETofEVENT]
DEPTH:	[depth: CARD8 visuals: LISTofVISUALTYPE]
VISUALTYPE:	[visual-id: VISUALID class: {**StaticGray**, **StaticColor**, **TrueColor**, **GrayScale**, **PseudoColor**, **DirectColor**} red-mask, green-mask, blue-mask: CARD32 bits-per-rgb-value: CARD8 colormap-entries: CARD16]

The information that is global to the server is:

The vendor string gives some identification of the owner of the server implementation. The vendor controls the semantics of the release number.

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

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

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

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

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

The server may retain the recent history of pointer motion and do so to a finer granularity than is reported by **MotionNotify** events. The **GetMotionEvents** request makes such history available. The motion-buffer-size gives the approximate maximum number of elements in the history buffer.

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

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

The information that applies per screen is:

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

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

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

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

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

Min-installed-maps specifies the number of maps that can be guaranteed to be installed simultaneously (with **InstallColormap**), regardless of the number of entries allocated in each map. Max-installed-maps specifies the maximum number of maps that might possibly be installed simultaneously, depending on their allocations. Multiple static-visual colormaps with identical contents but differing in resource ID should be considered as a single map for the purposes of this number. For the typical case of a single hardware colormap, both values will be 1.

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

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

The information that applies per visual-type is:

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 in the same way as **PseudoColor** except which primary drives the screen is undefined; thus, 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 in the same way as **DirectColor** except the colormap has predefined read-only RGB values. These values are server-dependent but provide linear or near-linear increasing ramps in each primary. **StaticColor** is treated in the same way as **PseudoColor** except the colormap has predefined read-only RGB values, which are server-dependent. **StaticGray** is treated in the same way 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 set to 1 with no intersections. Usually each mask has the same number of bits set to 1.

The bits-per-rgb-value specifies the log base 2 of the number of distinct color intensity values (individually) of red, green, and blue. This number need not bear any relation to the number of colormap entries. Actual RGB values are always passed in the protocol within a 16-bit spectrum, with 0 being minimum intensity and 65535 being the maximum intensity. On hardware that provides a linear zero-based intensity ramp, the following relationship exists:

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

Colormap entries are indexed from 0. The colormap-entries defines the number of available colormap entries in a newly created colormap. For **DirectColor** and **TrueColor**, this will usually be 2 to the power of the maximum number of bits set to 1 in red-mask, green-mask, and blue-mask.

9. Requests

CreateWindow

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

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

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

A class of **CopyFromParent** means the class is taken from the parent. A depth of zero for class **InputOutput** or **CopyFromParent** means the depth is taken from the parent. A visual of **CopyFromParent** means the visual type is taken from the parent. For class **InputOutput**, the visual type and depth must be a combination supported for the screen (or a **Match** error results). The depth need not be the same as the parent, but the parent must not be of class **InputOnly** (or a **Match** error results). For class **InputOnly**, the depth must be zero (or a **Match** error results), and the visual must be one supported for the screen (or a **Match** error results). However, the parent can 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 nonzero (or a **Value** error results). The border-width for an **InputOnly** window must be zero (or a **Match** error results).

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

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

The default values when attributes are not explicitly initialized are:

Attribute	Default
background-pixmap	**None**
border-pixmap	**CopyFromParent**
bit-gravity	**Forget**
win-gravity	**NorthWest**
backing-store	**NotUseful**
backing-planes	all ones

Attribute	Default
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
- override-redirect
- cursor

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

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

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

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

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** request).

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

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

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

The backing-planes indicates (with bits set to 1) which bit planes of the window hold dynamic data that must be preserved in backing-stores and during save-unders. The backing-pixel specifies what value to use in planes not covered by backing-planes. The server is free to save only the specified bit planes in the backing-store or save-under and regenerate the remaining planes with the specified pixel value. Any bits beyond the specified depth of the window in these values are simply ignored.

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

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

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

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

This request generates a **CreateNotify** event.

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

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

ChangeWindowAttributes

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

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

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

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

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

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

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

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

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

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

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

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

GetWindowAttributes

window: WINDOW

=>

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

Errors: **Window**

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

DestroyWindow

window: WINDOW

Errors: **Window**

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

Normal exposure processing on formerly obscured windows is performed.

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

DestroySubwindows

window: WINDOW

Errors: **Window**

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

ChangeSaveSet

window: WINDOW
mode: {**Insert**, **Delete**}

Errors: **Window**, **Match**, **Value**

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

When windows are destroyed, the server automatically removes them from the save-set.

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.

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

MapSubwindows

window: WINDOW

Errors: **Window**

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

UnmapWindow

window: WINDOW

Errors: **Window**

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

UnmapSubwindows

window: WINDOW

Errors: **Window**

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

ConfigureWindow

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

Errors: **Window**, **Match**, **Value**

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

Attribute	Type
x	INT16
y	INT16

Attribute	Type
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 nonzero (or a **Value** error results). Those values not specified are taken from the existing geometry of the window. Note that changing just the border-width leaves the outer-left corner of the window in a fixed position but moves the absolute position of the window's origin. It is a **Match** error to attempt to make the border-width of an **InputOnly** window nonzero.

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

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

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

If the inside width or height of a window is not changed but the window is moved or its border is changed, then the contents of the window are not lost but move with the window. Changing the inside width or height of the window causes its contents to be moved or lost, depending on the bit-gravity of the window. It also 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 as:

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

When a window with one of these bit-gravities is resized, the corresponding pair defines the change in position of each pixel in the window. When a window with one of these wingravities has its parent window resized, the corresponding pair defines the change in position of the window within the parent. This repositioning generates a **GravityNotify** event. **GravityNotify** events are generated after the **ConfigureNotify** event is generated.

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

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

The contents and borders of inferiors are not affected by their parent's bit-gravity. A server is permitted to ignore the specified bit-gravity and use **Forget** instead.

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

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

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

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

Above	The window is placed at the top of the stack.
Below	The window is placed at the bottom of the stack.
TopIf	If any sibling occludes the window, then the window is placed at the top of the stack.
BottomIf	If the window occludes any sibling, then the window is placed at the bottom of the stack.
Opposite	If any sibling occludes the window, then the window is placed at the top of the stack. Otherwise, if the window occludes any sibling, then the 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 to its initial geometry.

Attempts to configure a root window have no effect.

CirculateWindow

window: WINDOW
direction: {**RaiseLowest**, **LowerHighest**}

Errors: **Window**, **Value**

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

For **RaiseLowest**, **CirculateWindow** raises the lowest mapped child (if any) that is occluded by another child to the top of the stack. For **LowerHighest**, **CirculateWindow** lowers the highest mapped child (if any) that occludes another child to the bottom of the stack. Exposure processing is performed on formerly obscured windows.

GetGeometry

drawable: DRAWABLE

=>

root: WINDOW
depth: CARD8
x, y: INT16
width, height, border-width: CARD16

Errors: **Drawable**

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

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

QueryTree

window: WINDOW

=>

root: WINDOW
parent: WINDOW or **None**
children: LISTofWINDOW

Errors: **Window**

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

InternAtom

name: STRING8
only-if-exists: BOOL

=>

atom: ATOM or **None**

Errors: **Value**, **Alloc**

This request returns the atom for the given name. If only-if-exists is **False**, then the atom is created if it does not exist. The string should use the ISO Latin-1 encoding. Uppercase and lowercase matter.

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

GetAtomName

atom: ATOM

=>

name: STRING8

Errors: **Atom**

This request returns the name for the given atom.

ChangeProperty

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

Errors: **Window**, **Atom**, **Value**, **Match**, **Alloc**

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

If the 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 (or a **Match** error results). If the property is undefined, it is treated as defined with the correct type and format with zero-length data. For **Prepend**, the data is tacked on to the beginning of the existing data, and for **Append**, it is tacked on to the end of the existing data.

This request generates a **PropertyNotify** event on the window.

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

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

DeleteProperty

window: WINDOW
property: ATOM

Errors: **Window**, **Atom**

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

GetProperty

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

=>

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

Errors: **Window**, **Atom**, **Value**

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

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

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

RotateProperties

window: WINDOW
delta: INT16
properties: LISTofATOM

Errors: **Window**, **Atom**, **Match**

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

If delta mod N is nonzero, a **PropertyNotify** event is generated for each property in the order listed.

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

ListProperties

window: WINDOW

=>

atoms: LISTofATOM

Errors: **Window**

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

SetSelectionOwner

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

Errors: **Atom**, **Window**

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

If the client that is the owner of a selection is later terminated (that is, its connection is closed) or if the owner window it has specified in the request is later destroyed, then 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. The owner window is returned by the **GetSelectionOwner** request and is reported in **SelectionRequest** and **SelectionClear** events.

Selections are global to the server.

GetSelectionOwner

selection: ATOM

=>

owner: WINDOW or **None**

Errors: **Atom**

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

ConvertSelection

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

Errors: **Atom**, **Window**

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

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 and the focus window contains the pointer, destination is replaced with the window that the pointer is in. Otherwise, destination is replaced with the focus window.

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

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

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

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

Active grabs are ignored for this request.

GrabPointer

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

=>

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

Errors: **Cursor**, **Window**, **Value**

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

If owner-events is **False**, all generated pointer events are reported with respect to grab-window and are only reported if selected by event-mask. If owner-events is **True** and a generated pointer event would normally be reported to this client, it is reported normally. Otherwise, the event is reported with respect to the grab-window and is only reported if selected by event-mask. For either value of owner-events, unreported events are simply discarded.

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

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

If a cursor is specified, then it is displayed regardless of what window the pointer is in. If no cursor is specified, then when the pointer is in grab-window or one of its subwindows, the normal cursor for that window is displayed. 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 are 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 or if the confine-to window lies completely outside the boundaries of the root window. The request fails with status **InvalidTime** if the specified time is earlier than the last-pointer-grab time or later than the current server time. Otherwise, the last-pointer-grab time is set to the specified time, with **CurrentTime** replaced by the current server time.

UngrabPointer

time: TIMESTAMP or **CurrentTime**

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

This request generates **EnterNotify** and **LeaveNotify** events.

An **UngrabPointer** request is performed automatically if the event window or confine-to window for an active pointer grab becomes not viewable or if window reconfiguration causes the confine-to window to lie completely outside the boundaries of the root window.

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, 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 if all of the following conditions are true:

- The pointer is not grabbed and the specified button is logically pressed when the specified modifier keys are logically down, and no other buttons or modifier keys are logically down.
- The grab-window contains the pointer.
- The confine-to window (if any) is viewable.
- A passive grab on the same button/key combination does not exist on any ancestor of grab-window.

The interpretation of the remaining arguments is the same as for **GrabPointer**. The active grab is terminated automatically when the logical state of the pointer has all buttons released, independent of the logical state of modifier keys. Note that the logical state of a device (as seen by means of the protocol) may lag the physical state if device event processing is frozen.

This request overrides all previous passive grabs by the same client on the same button/key combinations on the same window. A modifier of **AnyModifier** is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all specified modifiers have currently assigned keycodes. A button of **AnyButton** is equivalent to issuing the request for all possible buttons. Otherwise, it is not required that the button specified currently be assigned to a physical button.

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

UngrabButton

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

Errors: **Window**, **Value**

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

ChangeActivePointerGrab

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

Errors: **Cursor**, **Value**

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

GrabKeyboard

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

=>

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

Errors: **Window**, **Value**

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

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

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

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

This request generates **FocusIn** and **FocusOut** events.

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

UngrabKeyboard

time: TIMESTAMP or **CurrentTime**

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

This request generates **FocusIn** and **FocusOut** events.

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

GrabKey

key: KEYCODE or **AnyKey**
modifiers: SETofKEYMASK or **AnyModifier**
grab-window: WINDOW
owner-events: BOOL
pointer-mode, *keyboard-mode*: {**Synchronous**, **Asynchronous**}

Errors: **Window**, **Value**, **Access**

This request establishes a passive grab on the keyboard. In the future, the keyboard is actively grabbed as described in **GrabKeyboard**, the last-keyboard-grab time is set to the time at which the key was pressed (as transmitted in the **KeyPress** event), and the **KeyPress** event is reported if all of the following conditions are true:

- The keyboard is not grabbed and the specified key (which can itself be a modifier key) is logically pressed when the specified modifier keys are logically down, and no other modifier keys are logically down.
- Either the grab-window is an ancestor of (or is) the focus window, or the grab-window is a descendent of the focus window and contains the pointer.
- A passive grab on the same key combination does not exist on any ancestor of grab-window.

The interpretation of the remaining arguments is the same as for **GrabKeyboard**. The active grab is terminated automatically when the logical state of the keyboard has the specified key released, independent of the logical state of modifier keys. Note that the logical state of a device (as seen by means of the protocol) may lag the physical state if device event processing is frozen.

This request overrides all previous passive grabs by the same client on the same key combinations on the same window. A modifier of **AnyModifier** is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned keycodes. A key of **AnyKey** is equivalent to issuing the request for all possible keycodes. Otherwise, the key must be in the range specified by min-keycode and max-keycode in the connection setup (or a **Value** error results).

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

UngrabKey

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

Errors: **Window**, **Value**

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

AllowEvents

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

Errors: **Value**

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

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

For **SyncPointer**, if the pointer is frozen and actively grabbed by the client, pointer event processing continues normally until the next **ButtonPress** or **ButtonRelease** event is reported to the client, at which time the pointer again appears to freeze. However, if the reported event causes the pointer grab to be released, then the pointer does not freeze. **SyncPointer** has no effect if the pointer is not frozen by the client or if the pointer is not grabbed by the client.

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

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

For **SyncKeyboard**, if the keyboard is frozen and actively grabbed by the client, keyboard event processing continues normally until the next **KeyPress** or **KeyRelease** event is reported to the client, at which time the keyboard again appears to freeze. However, if the reported event causes the keyboard grab to be released, then the keyboard does not freeze. **SyncKeyboard** has no effect if the keyboard is not frozen by the client or if the keyboard is not grabbed by the client.

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

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

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

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

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

GrabServer

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

UngrabServer

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

QueryPointer

window: WINDOW

=>

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

Errors: **Window**

The root window the pointer is logically on and the 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, child is **None**, and win-x and win-y are zero. If same-screen is **True**, then win-x and win-y are the pointer coordinates relative to the argument window's origin, and child is the child containing the pointer, if any. The current logical state of the modifier keys and the buttons are also returned. Note that the logical state of a device (as seen by means of the protocol) may lag the physical state if device event processing is frozen.

GetMotionEvents

start, *stop*: TIMESTAMP or **CurrentTime**
window: WINDOW

=>

events: LISTofTIMECOORD

where:

TIMECOORD: [x, y: INT16
time: TIMESTAMP]

Errors: **Window**

This request returns all events in the motion history buffer that fall between the specified start and stop times (inclusive) and that have coordinates that lie within (including borders) the specified window at its present placement. The x and y coordinates are reported relative to the origin of the window.

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

TranslateCoordinates

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

=>

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

Errors: **Window**

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

WarpPointer

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

Errors: **Window**

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

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

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

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

SetInputFocus

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

Errors: **Window**, **Value**, **Match**

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

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

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

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

This request generates **FocusIn** and **FocusOut** events.

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

GetInputFocus

=>

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

This request returns the current focus state.

QueryKeymap

=>

keys: LISTofCARD8

This request returns a bit vector for the logical state of the keyboard. Each bit set to 1 indicates that the corresponding key is currently pressed. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7 with the least-significant bit in the byte representing key 8N. Note that the logical state of a device (as seen by means of the protocol) may lag the physical state if device event processing is frozen.

OpenFont

fid: FONT
name: STRING8

Errors: **IDChoice**, **Name**, **Alloc**

This request loads the specified font, if necessary, and associates identifier fid with it. The font name should use the ISO Latin-1 encoding, and uppercase and lowercase do not matter. The interpretation of characters ‘‘?’’ (octal value 77) and ‘‘*’’ (octal value 52) in the name is not defined by the core protocol, but is reserved for future definition. A structured format for font names is specified in the X Consortium standard ‘‘X Logical Font Description Conventions’’.

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

CloseFont

font: FONT

Errors: **Font**

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

QueryFont

font: FONTABLE

=>

font-info: FONTINFO
char-infos: LISTofCHARINFO

where:

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

Errors: **Font**

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

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

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

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

where:

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

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

L = D * (max-byte1 – min-byte1 + 1)

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

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

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

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

with a width of:

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

and a height of:

max-bounds.ascent + max-bounds.descent

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

A font is not guaranteed to have any properties. The interpretation of the property value (for example, INT32, CARD32) must be derived from a priori knowledge of the property. A basic set of font properties is specified in the X Consortium standard ‘‘X Logical Font Description Conventions’’.

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

[x + left-side-bearing, y – ascent]

with a width of:

right-side-bearing – left-side-bearing

and a height of:

ascent + descent

and the origin for the next character is defined to be:

[x + character-width, y]

Note that the baseline is logically viewed as being just below nondescending 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 nonkerned 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 nonexistent character is represented with all CHARINFO components zero.

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

QueryTextExtents

font: FONTABLE
string: STRING16

=>

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

Errors: **Font**

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

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

Characters with all zero metrics are ignored. If the font has no defined default-char, then undefined characters in the string are also ignored.

ListFonts

pattern: STRING8
max-names: CARD16

=>

names: LISTofSTRING8

This request returns a list of available font names (as controlled by the font search path; see **SetFontPath** request) that match the pattern. At most, max-names names will be returned. The pattern should use the ISO Latin-1 encoding, and uppercase and lowercase do not matter. In the pattern, the ‘‘?’’ character (octal value 77) will match any single character, and the ‘‘*’’ character (octal value 52) will match any number of characters. The returned names are in lowercase.

ListFontsWithInfo

pattern: STRING8
max-names: CARD16

=>+

name: STRING8
info: FONTINFO
replies-hint: CARD32

where:

FONTINFO: <same type definition as in **QueryFont**>

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

SetFontPath

path: LISTofSTRING8

Errors: **Value**

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

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

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

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

GetFontPath

=>

path: LISTofSTRING8

This request returns the current search path for fonts.

CreatePixmap

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

Errors: **IDChoice**, **Drawable**, **Value**, **Alloc**

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

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

FreePixmap

pixmap: PIXMAP

Errors: **Pixmap**

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

CreateGC

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

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

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

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

Component	Type
function	{**Clear**, **And**, **AndReverse**, **Copy**, **AndInverted**, **NoOp**, **Xor**, **Or**, **Nor**, **Equiv**, **Invert**, **OrReverse**, **CopyInverted**, **OrInverted**, **Nand**, **Set**}
plane-mask	CARD32
foreground	CARD32
background	CARD32
line-width	CARD16
line-style	{**Solid**, **OnOffDash**, **DoubleDash**}
cap-style	{**NotLast**, **Butt**, **Round**, **Projecting**}
join-style	{**Miter**, **Round**, **Bevel**}
fill-style	{**Solid**, **Tiled**, **OpaqueStippled**, **Stippled**}
fill-rule	{**EvenOdd**, **Winding**}
arc-mode	{**Chord**, **PieSlice**}
tile	PIXMAP
stipple	PIXMAP
tile-stipple-x-origin	INT16
tile-stipple-y-origin	INT16
font	FONT
subwindow-mode	{**ClipByChildren**, **IncludeInferiors**}
graphics-exposures	BOOL
clip-x-origin	INT16
clip-y-origin	INT16
clip-mask	PIXMAP or **None**
dash-offset	CARD16
dashes	CARD8

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

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

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

The meanings of the functions are:

Function	Operation

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

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

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

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

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

Thin lines (zero line-width) are ‘‘one pixel wide’’ lines drawn using an unspecified, device-dependent algorithm. There are only two constraints on this algorithm. First, if a line is drawn unclipped from [x1,y1] to [x2,y2] and another line is drawn unclipped from [x1+dx,y1+dy] to [x2+dx,y2+dy], then a point [x,y] is touched by drawing the first line if and only if the point [x+dx,y+dy] is touched by drawing the second line. Second, the effective set of points comprising a line cannot be affected by clipping. Thus, 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-style and join-style. Implementors are encouraged to make this property true for thin lines, but it is not required. A line-width of zero may differ from a line-width of one in which pixels are drawn. In general, drawing a thin line will be faster than drawing a wide line of width one, but thin lines may not mix well aesthetically with wide lines because of the different drawing algorithms. If it is desirable to obtain precise and uniform results across all displays, a client should always use a line-width of one, rather than a line-width of zero.

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

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

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

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

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. However, if the angle is less than 11 degrees, a **Bevel** join-style is used instead.
Round	The result is a circular arc with a diameter equal to the line-width, centered on the joinpoint.
Bevel	The result is **Butt** endpoint styles, and then the triangular ‘‘notch’’ is filled.

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

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

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

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

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

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

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

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

Solid	Background
Tiled	Same as for even dashes
OpaqueStippled	Same as for even dashes
Stippled	Background masked by stipple

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

The clip-mask restricts writes to the destination drawable. Only pixels where the clip-mask has bits set to 1 are drawn. Pixels are not drawn outside the area covered by the clip-mask or where the clip-mask has bits set to 0. The clip-mask affects all graphics requests, but it does not clip sources. The clip-mask origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. If a pixmap is specified as the clip-mask, it must have depth 1 and have the same root as the gcontext (or a **Match** error results). If clip-mask is **None**, then pixels are always drawn, regardless of the clip origin. The clip-mask can also be set with the **SetClipRectangles** request.

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

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

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

The arc-mode controls filling in the **PolyFillArc** request.

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

The default component values are:

Component	Default
function	**Copy**
plane-mask	all ones
foreground	0
background	1
line-width	0
line-style	**Solid**
cap-style	**Butt**
join-style	**Miter**
fill-style	**Solid**
fill-rule	**EvenOdd**
arc-mode	**PieSlice**
tile	Pixmap of unspecified size filled with foreground pixel (that is, client specified pixel if any, else 0) (subsequent changes to foreground do not affect this pixmap)
stipple	Pixmap of unspecified size filled with ones
tile-stipple-x-origin	0
tile-stipple-y-origin	0
font	<server-dependent-font>
subwindow-mode	**ClipByChildren**
graphics-exposures	**True**
clip-x-origin	0
clip-y-origin	0
clip-mask	**None**
dash-offset	0
dashes	4 (that is, the list [4, 4])

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

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

ChangeGC

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

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

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

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

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

CopyGC

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

Errors: **GContext**, **Value**, **Match**, **Alloc**

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

SetDashes

gc: GCONTEXT
dash-offset: CARD16
dashes: LISTofCARD8

Errors: **GContext**, **Value**, **Alloc**

This request sets dash-offset and dashes in gc for dashed line styles. Dashes cannot be empty (or a **Value** error results). Specifying an odd-length list is equivalent to specifying the same list concatenated with itself to produce an even-length list. The initial and alternating elements of dashes are the even dashes; the others are the odd dashes. Each element specifies a dash length in pixels. All of the elements must be nonzero (or a **Value** error results). The dash-offset defines the phase of the pattern, specifying how many pixels into dashes the pattern should actually begin in any single graphics request. Dashing is continuous through path elements combined with a join-style, but is reset to the dash-offset between each sequence of joined lines.

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

SetClipRectangles

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

Errors: **GContext**, **Value**, **Alloc**, **Match**

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

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

FreeGC

gc: GCONTEXT

Errors: **GContext**

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

ClearArea

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

Errors: **Window**, **Value**, **Match**

The x and y coordinates are relative to the window's origin and specify the upper-left corner of the rectangle. If width is zero, it is replaced with the current width of the window minus x. If height is zero, it is replaced with the current height of the window minus y. If the window has a defined background tile, the rectangle is tiled with a plane-mask of all ones and function of **Copy** and a subwindow-mode of **ClipByChildren**. If the window has background **None**, the contents of the window are not changed. In either case, if exposures is **True**, then one or more exposure events are generated for regions of the rectangle that are either visible or are being retained in a backing store.

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

CopyArea

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

Errors: **Drawable**, **GContext**, **Match**

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

If regions of the source rectangle are obscured and have not been retained in backing store or if regions outside the boundaries of the source drawable are specified, then those regions are not copied, but the following occurs on all corresponding destination regions that are either visible or are retained in backing-store. If the dst-drawable is a window with a background other than **None**, these corresponding destination regions are tiled (with plane-mask of all ones and function **Copy**) with that background. Regardless of tiling and whether the destination is a window or a pixmap, if graphics-exposures in gc is **True**, then **GraphicsExposure** events for all corresponding destination regions are generated.

If graphics-exposures is **True** but no **GraphicsExposure** events are generated, then a **NoExposure** event is generated.

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

CopyPlane

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

Errors: **Drawable**, **GContext**, **Value**, **Match**

The src-drawable must have the same root as dst-drawable (or a **Match** error results), but it need not have the same depth. The bit-plane must have exactly one bit set to 1 and the value of bit-plane must be less than 2^n where n is the depth of src-drawable (or a **Value** error results). Effectively, a pixmap of the same depth as dst-drawable and with size specified by the source region is formed using the foreground/background pixels in gc (foreground everywhere the bit-plane in src-drawable contains a bit set to 1, background everywhere the bit-plane contains a bit set to 0), and the equivalent of a **CopyArea** is performed, with all the same exposure semantics. This can also be thought of as using the specified region of the source bit-plane as a stipple with a fill-style of **OpaqueStippled** for filling a rectangular area of the destination.

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

PolyPoint

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

Errors: **Drawable**, **GContext**, **Value**, **Match**

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

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

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

PolyLine

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

Errors: **Drawable**, **GContext**, **Value**, **Match**

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

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

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

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

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

PolySegment

drawable: DRAWABLE
gc: GCONTEXT
segments: LISTofSEGMENT

where:

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

Errors: **Drawable**, **GContext**, **Match**

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

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

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

PolyRectangle

drawable: DRAWABLE
gc: GCONTEXT
rectangles: LISTofRECTANGLE

Errors: **Drawable**, **GContext**, **Match**

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

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

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

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

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

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

PolyArc

drawable: DRAWABLE
gc: GCONTEXT

arcs: LISTofARC

Errors: **Drawable**, **GContext**, **Match**

This request draws circular or elliptical arcs. Each arc is specified by a rectangle and two angles. The angles are signed integers in degrees scaled by 64, with positive indicating counterclockwise motion and negative indicating clockwise motion. The start of the arc is specified by angle1 relative to the three-o'clock position from the center of the rectangle, and the path and extent of the arc is specified by angle2 relative to the start of the arc. If the magnitude of angle2 is greater than 360 degrees, it is truncated to 360 degrees. The x and y coordinates of the rectangle are relative to the origin of the drawable. For an arc specified as [x,y,w,h,a1,a2], the origin of the major and minor axes is at [x+(w/2),y+(h/2)], and the infinitely thin path describing the entire circle/ellipse intersects the horizontal axis at [x,y+(h/2)] and [x+w,y+(h/2)] and intersects the vertical axis at [x+(w/2),y] and [x+(w/2),y+h]. These coordinates can be fractional; that is, they are not truncated to discrete coordinates. The path should be defined by the ideal mathematical path. For a wide line with line-width lw, the bounding outlines for filling are given by the two infinitely thin paths consisting of all points whose perpendicular distance from the path of the circle/ellipse is equal to lw/2 (which may be a fractional value). The cap-style and join-style are applied the same as for a line corresponding to the tangent of the circle/ellipse at the endpoint.

For an arc specified as [x,y,w,h,a1,a2], the angles must be specified in the effectively skewed coordinate system of the ellipse (for a circle, the angles and coordinate systems are identical). The relationship between these angles and angles expressed in the normal coordinate system of the screen (as measured with a protractor) is as follows:

skewed-angle = atan(tan(normal-angle) * w/h) + adjust

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

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

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

By specifying one axis to be zero, a horizontal or vertical line can be drawn.

Angles are computed based solely on the coordinate system, ignoring the aspect ratio.

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

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

FillPoly

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

points: LISTofPOINT

Errors: **Drawable**, **GContext**, **Match**, **Value**

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

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

The shape parameter may be used by the server to improve performance. **Complex** means the path may self-intersect. Contiguous coincident points in the path are not treated as self-intersection.

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 that for every pair of points inside the polygon, the line segment connecting them does not intersect the path. If known by the client, specifying **Convex** can improve performance. If **Convex** is specified for a path that is not convex, the graphics results are undefined.

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

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

PolyFillRectangle

drawable: DRAWABLE
gc: GCONTEXT
rectangles: LISTofRECTANGLE

Errors: **Drawable**, **GContext**, **Match**

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

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

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

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

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

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

PolyFillArc

drawable: DRAWABLE
gc: GCONTEXT
arcs: LISTofARC

Errors: **Drawable**, **GContext**, **Match**

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

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

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

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

PutImage

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

Errors: **Drawable**, **GContext**, **Match**, **Value**

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

If **Bitmap** format is used, then depth must be one (or a **Match** error results), and the image must be in XY format. The foreground pixel in gc defines the source for bits set to 1 in the image, and the background pixel defines the source for the bits set to 0.

For **XYPixmap** and **ZPixmap**, the depth must match the depth of the drawable (or a **Match** error results). For **XYPixmap**, the image must be sent in XY format. For **ZPixmap**, the image must be sent in the Z format defined for the given depth.

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

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

GC mode-dependent components: foreground, background

GetImage

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

=>

depth: CARD8
visual: VISUALID or **None**
data: LISTofBYTE

Errors: **Drawable**, **Value**, **Match**

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

If the drawable is a pixmap, then the given rectangle must be wholly contained within the pixmap (or a **Match** error results). If the drawable is a window, the window must be viewable, and it must be the case that, if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen and wholly contained within the outside edges of the window (or a **Match** error results). Note that the borders of the window can be included and read with this request. If the window has a backing store, then the backing-store contents are returned for regions of the window that are obscured by noninferior windows; otherwise, the returned contents of such obscured regions are undefined. Also undefined are the returned contents of visible regions of inferiors of different depth than the specified window. The pointer cursor image is not included in the contents returned.

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

PolyText8

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

where:

TEXTITEM8: TEXTELT8 or FONT
TEXTELT8: [delta: INT8
string: STRING8]

Errors: **Drawable**, **GContext**, **Match**, **Font**

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

All contained FONTs are always transmitted most-significant byte first.

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

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

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

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

PolyText16

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

where:

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

Errors: **Drawable**, **GContext**, **Match**, **Font**

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

ImageText8

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

Errors: **Drawable**, **GContext**, **Match**

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

[x, y – font-ascent]

the width is:

overall-width

and the height is:

font-ascent + font-descent

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

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

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

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

ImageText16

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

Errors: **Drawable**, **GContext**, **Match**

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

CreateColormap

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

Errors: **IDChoice**, **Window**, **Value**, **Match**, **Alloc**

This request creates a colormap of the specified visual type for the screen on which the window resides and associates the identifier mid with it. The visual type must be one supported by the screen (or a **Match** error results). 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** (or a **Match** error results). For the other classes, if alloc is **None**, the colormap initially has no allocated entries, and clients can allocate entries.

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

FreeColormap

cmap: COLORMAP

Errors: **Colormap**

This request deletes the association between the resource ID and the colormap and frees the colormap storage. If the colormap is an installed map for a screen, it is uninstalled (see **UninstallColormap** request). If the colormap is defined as the colormap for a window (by means of **CreateWindow** or **ChangeWindowAttributes**), the colormap for the window is changed to **None**, and a **ColormapNotify** event is generated. The protocol does not define the colors displayed for a window with a colormap of **None**.

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

CopyColormapAndFree

mid, src-cmap: COLORMAP

Errors: **IDChoice**, **Colormap**, **Alloc**

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

InstallColormap

cmap: COLORMAP

Errors: **Colormap**

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

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

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

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

UninstallColormap

cmap: COLORMAP

Errors: **Colormap**

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

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

ListInstalledColormaps

window: WINDOW

=>

cmaps: LISTofCOLORMAP

Errors: **Window**

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

AllocColor

cmap: COLORMAP
red, green, blue: CARD16

=>

pixel: CARD32
red, green, blue: CARD16

Errors: **Colormap**, **Alloc**

This request allocates a read-only colormap entry corresponding to the closest RGB values provided by the hardware. It also returns the pixel and the RGB values actually used. Multiple clients requesting the same effective RGB values can be assigned the same read-only entry, allowing entries to be shared.

AllocNamedColor

cmap: COLORMAP
name: STRING8

=>

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

Errors: **Colormap**, **Name**, **Alloc**

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

AllocColorCells

cmap: COLORMAP
colors, *planes*: CARD16
contiguous: BOOL

=>

pixels, masks: LISTofCARD32

Errors: **Colormap**, **Value**, **Alloc**

The number of colors must be positive, and the number of planes must be nonnegative (or a **Value** error results). 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 set to 1; for **DirectColor**, each will have exactly three bits set to 1. If contiguous is **True** and if all masks are ORed together, a single contiguous set of bits will be formed for **GrayScale** or **PseudoColor**, and three contiguous sets of bits (one within each pixel subfield) for **DirectColor**. The RGB values of the allocated entries are undefined.

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

FreeColors

cmap: COLORMAP
pixels: LISTofCARD32
plane-mask: CARD32

Errors: **Colormap**, **Access**, **Value**

The plane-mask should not have any bits in common with any of the pixels. The set of all pixels is produced by ORing together subsets of plane-mask with the pixels. The request frees all of these pixels that were allocated by the client (using **AllocColor**, **AllocNamedColor**, **AllocColorCells**, and **AllocColorPlanes**). Note that freeing an individual pixel obtained from **AllocColorPlanes** may not actually allow it to be reused until all of its related pixels are also freed. Similarly, a read-only entry is not actually freed until it has been freed by all clients, and if a client allocates the same read-only entry multiple times, it must free the entry that many times before the entry is actually freed.

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

StoreColors

cmap: COLORMAP
items: LISTofCOLORITEM

where:

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

Errors: **Colormap**, **Access**, **Value**

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

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

StoreNamedColor

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

Errors: **Colormap**, **Name**, **Access**, **Value**

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

QueryColors

cmap: COLORMAP
pixels: LISTofCARD32

=>

colors: LISTofRGB

where:

RGB: [red, green, blue: CARD16]

Errors: **Colormap**, **Value**

This request returns the hardware-specific 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, it is arbitrary as to which pixel is reported.

LookupColor

cmap: COLORMAP
name: STRING8

=>

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

Errors: **Colormap**, **Name**

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

CreateCursor

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

Errors: **IDChoice**, **Pixmap**, **Match**, **Alloc**

This request creates a cursor and associates identifier cid with it. The foreground and background RGB values must be specified, even if the server only has a **StaticGray** or **GrayScale** screen. The foreground is used for the bits set to 1 in the source, and the background is used for the bits set to 0. Both source and mask (if specified) must have depth one (or a **Match** error results), but they can have any root. The mask pixmap defines the shape of the cursor. That is, the bits set to 1 in the mask define which source pixels will be displayed, and where the mask has bits set to 0, the corresponding bits of the source pixmap are ignored. If no mask is given, all pixels of the source are displayed. The mask, if present, must be the same size as the source (or a **Match** error results). The x and y coordinates define the hotspot relative to the source's origin and must be a point within the source (or a **Match** error results).

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

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

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

CreateGlyphCursor

cid: CURSOR
source-font: FONT
mask-font: FONT or **None**

source-char, *mask-char*: CARD16
fore-red, *fore-green*, *fore-blue*: CARD16
back-red, *back-green*, *back-blue*: CARD16

Errors: **IDChoice**, **Font**, **Value**, **Alloc**

This request is similar to **CreateCursor**, except the source and mask bitmaps are obtained from the specified font glyphs. The source-char must be a defined glyph in source-font, and if mask-font is given, mask-char must be a defined glyph in mask-font (or a **Value** error results). The mask font and character are optional. The origins of the source and mask (if it is defined) glyphs are positioned coincidently and define the hotspot. The source and mask need not have the same bounding box metrics, and there is no restriction on the placement of the hotspot relative to the bounding boxes. If no mask is given, all pixels of the source are displayed. Note that source-char and mask-char are CARD16, not CHAR2B. For 2-byte matrix fonts, the 16-bit value should be formed with byte1 in the most-significant byte and byte2 in the least-significant byte.

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

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

FreeCursor

cursor: CURSOR

Errors: **Cursor**

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

RecolorCursor

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

Errors: **Cursor**

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

QueryBestSize

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

=>

width, height: CARD16

Errors: **Drawable**, **Value**, **Match**

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

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

QueryExtension

name: STRING8

=>

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

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

The extension name should use the ISO Latin-1 encoding, and uppercase and lowercase matter.

ListExtensions

=>

names: LISTofSTRING8

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

SetModifierMapping

keycodes-per-modifier: CARD8
keycodes: LISTofKEYCODE

=>

status: {**Success**, **Busy**, **Failed**}

Errors: **Value**, **Alloc**

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

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

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

This request generates a **MappingNotify** event on a **Success** status.

GetModifierMapping

=>

keycodes-per-modifier: CARD8
keycodes: LISTofKEYCODE

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

ChangeKeyboardMapping

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

Errors: **Value**, **Alloc**

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

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

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

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

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

This request generates a **MappingNotify** event.

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

GetKeyboardMapping

first-keycode: KEYCODE
count: CARD8

=>

keysyms-per-keycode: CARD8
keysyms: LISTofKEYSYM

Errors: **Value**

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

first-keycode + count – 1

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

count * keysyms-per-keycode

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

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

in keysyms. The keysyms-per-keycode value is chosen arbitrarily by the server to be large enough to report all requested symbols. A special KEYSYM value of **NoSymbol** is used to fill in unused elements for individual keycodes.

ChangeKeyboardControl

value-mask: BITMASK
value-list: LISTofVALUE

Errors: **Match**, **Value**

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

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

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

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

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

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

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

If both auto-repeat-mode and key are specified, then the auto-repeat mode of that key is changed, if possible. If only auto-repeat-mode is specified, then the global auto-repeat mode for the entire keyboard is changed, if possible, without affecting the per-key settings. It is a **Match** error if a key is specified without an auto-repeat-mode. Each key has an individual mode of whether or not it should auto-repeat and a default setting for that mode. In addition, there is a global mode of whether auto-repeat should be enabled or not and a default setting for that mode. When the global mode is **On**, keys should obey their individual auto-repeat modes. When the global mode is **Off**, no keys should auto-repeat. An auto-repeating key generates alternating **KeyPress** and **KeyRelease** events. When a key is used as a modifier, it is desirable for the key not to auto-repeat, regardless of the auto-repeat setting for that key.

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

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

GetKeyboardControl

=>

key-click-percent: CARD8
bell-percent: CARD8

bell-pitch: CARD16
bell-duration: CARD16
led-mask: CARD32
global-auto-repeat: {**On**, **Off**}
auto-repeats: LISTofCARD8

This request returns the current control values for the keyboard. For the LEDs, the least-significant bit of led-mask corresponds to LED one, and each one bit in led-mask indicates an LED that is lit. The auto-repeats is a bit vector; each one bit indicates that auto-repeat is enabled for the corresponding key. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7, with the least-significant bit in the byte representing key 8N.

Bell

percent: INT8

Errors: **Value**

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

$$\text{base} - [(\text{base} * \text{percent}) / 100] + \text{percent}$$

When percent is negative, it is:

$$\text{base} + [(\text{base} * \text{percent}) / 100]$$

SetPointerMapping

map: LISTofCARD8

=>

status: {**Success**, **Busy**}

Errors: **Value**

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

A zero element disables a button. Elements are not restricted in value by the number of physical buttons, but no two elements can have the same nonzero value (or a **Value** error results).

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

This request generates a **MappingNotify** event on a **Success** status.

GetPointerMapping

=>

map: LISTofCARD8

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

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

ChangePointerControl

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

Errors: **Value**

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

GetPointerControl

=>

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

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

SetScreenSaver

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

Errors: **Value**

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

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

GetScreenSaver

=>

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

This request returns the current screen-saver control values.

ForceScreenSaver

mode: {**Activate**, **Reset**}

Errors: **Value**

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

ChangeHosts

mode: {**Insert**, **Delete**}
host: HOST

Errors: **Access**, **Value**

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

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

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

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

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

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

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

ListHosts

=>

mode: {**Enabled**, **Disabled**}
hosts: LISTofHOST

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

Each HOST is padded to a multiple of four bytes.

SetAccessControl

mode: {**Enable**, **Disable**}

Errors: **Value**, **Access**

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

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

SetCloseDownMode

mode: {**Destroy**, **RetainPermanent**, **RetainTemporary**}

Errors: **Value**

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

KillClient

resource: CARD32 or **AllTemporary**

Errors: **Value**

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

NoOperation

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

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

10. Connection Close

At connection close, all event selections made by the client are discarded. If the client has the pointer actively grabbed, an **UngrabPointer** is performed. If the client has the keyboard actively grabbed, an **UngrabKeyboard** is performed. All passive grabs by the client are released. If the client has the server grabbed, an **UngrabServer** is performed. All selections (see **SetSelectionOwner** request) owned by the client are disowned. If close-down mode (see **SetCloseDownMode** request) 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**, all of the client's resources are destroyed.

When a client's resources are destroyed, for each window in the client's save-set, if the window is an inferior of a window created by the client, the save-set window is reparented to the closest ancestor such that the save-set window is not an inferior of a window created by the client. If the save-set window is unmapped, a **MapWindow** request is performed on it (even if it was not an inferior of a window created by the client). The reparenting leaves unchanged the absolute coordinates (with respect to the root window) of the upper-left outer corner of the save-set window. After save-set processing, all windows created by the client are destroyed. For each nonwindow resource created by the client, the appropriate **Free** request is performed. All colors and colormap entries allocated by the client are freed.

A server goes through a cycle of having no connections and having some connections. At every transition to the state of having no connections as a result of a connection closing with a **Destroy** close-down mode, the server resets its state as if it had just been started. This starts by destroying all lingering resources from clients that have terminated in **RetainPermanent** or **RetainTemporary** mode. It additionally includes deleting all but the predefined atom identifiers, deleting all properties on all root windows, resetting all device maps and attributes (key click, bell volume, acceleration), resetting the access control list, restoring the standard root tiles and cursors, restoring the default font path, and restoring the input focus to state **PointerRoot**.

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

11. Events

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

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

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

KeyPress
KeyRelease
ButtonPress
ButtonRelease
MotionNotify

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

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

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

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

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

EnterNotify
LeaveNotify

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

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

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

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

Normal events are generated as follows:

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

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

- **EnterNotify** with detail **Inferior** is generated on B.

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

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

When the pointer moves from window A to window B and window C is their least common ancestor:

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

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

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

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

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

When a pointer grab deactivates (but after generating any actual **ButtonRelease** event that deactivates the grab), G is the grab-window for the grab, and P is 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 some position in G to its current position in P. However, the pointer does not warp, and the current pointer position is used as both the initial and final positions for the events.

FocusIn
FocusOut

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

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

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

Normal and **WhileGrabbed** events are generated as follows:

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

- If P is an inferior of A but P is not 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 order).
- **FocusIn** with detail **Ancestor** is generated on B.

When the focus moves from window A to window B, window C is their least common ancestor, and the pointer is 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 order).
- **FocusIn** with detail **NonlinearVirtual** is generated on each window between C and B exclusive (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 window B on different screens and the pointer is in window P:

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

When the focus moves from window A to **PointerRoot** (or **None**) and the pointer is in window P:

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

When the focus moves from **PointerRoot** (or **None**) to window A and the pointer is in window P:

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

When the focus moves from **PointerRoot** to **None** (or vice versa) and the pointer is in window P:

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

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

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

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

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

KeymapNotify

keys: LISTofCARD8

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

Expose

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

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

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

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

Expose events are never generated on **InputOnly** windows.

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

GraphicsExposure

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

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

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

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

NoExposure

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

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

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

VisibilityNotify

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

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

VisibilityNotify events are never generated on **InputOnly** windows.

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

CreateNotify

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

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

DestroyNotify

event, *window*: WINDOW

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

The ordering of the **DestroyNotify** events is such that for any given window, **DestroyNotify** is generated on all inferiors of the window before being generated on the window itself. The ordering among siblings and across subhierarchies is not otherwise constrained.

UnmapNotify

event, *window*: WINDOW
from-configure: BOOL

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

MapNotify

event, *window*: WINDOW
override-redirect: BOOL

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

MapRequest

parent, *window*: WINDOW

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

ReparentNotify

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

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

ConfigureNotify

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

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

GravityNotify

event, window: WINDOW
x, y: INT16

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

ResizeRequest

window: WINDOW
width, height: CARD16

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

ConfigureRequest

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

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

CirculateNotify

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

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

CirculateRequest

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

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

PropertyNotify

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

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

SelectionClear

owner: WINDOW
selection: ATOM
time: TIMESTAMP

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

SelectionRequest

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

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

SetSelectionOwner request. The remaining arguments are as in the **ConvertSelection** request.

The owner should convert the selection based on the specified target type and send a **SelectionNotify** back to the requestor. A complete specification for using selections is given in the X Consortium standard ‘‘Inter-Client Communication Conventions Manual’’.

SelectionNotify

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

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

ColormapNotify

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

This event is reported to clients selecting **ColormapChange** on the window. It is generated with value **True** for new when the colormap attribute of the window is changed and is generated with value **False** for new when the colormap of a window is installed or uninstalled. In either case, the state indicates whether the colormap is currently installed.

MappingNotify

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

This event is sent to all clients. There is no mechanism to express disinterest in this event. The detail indicates the kind of change that occurred: **Modifiers** for a successful **SetModifierMapping**, **Keyboard** for a successful **ChangeKeyboardMapping**, and **Pointer** for a successful **SetPointerMapping**. If the detail is **Keyboard**, then first-keycode and count indicate the range of altered keycodes.

ClientMessage

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

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

12. Flow Control and Concurrency

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

Whether or not 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 requests from a given connection must be executed in delivery order (that is, the total execution order is a shuffle of the individual streams). The execution of a request includes validating all arguments, collecting all data for any reply, and generating and queueing all required events. However, it does not include the actual transmission of the reply and the events. In addition, the effect of any other cause that can generate multiple events (for example, activation of a grab or pointer motion) must effectively generate and queue all required events indivisibly with respect to all other causes and requests. For a request from a given client, any events destined for that client that are caused by executing the request must be sent to the client before any reply or error is sent.

Table of Contents