X Window System

Version 11

Release 5

X Window System Protocol

Xlib - C Language X Interface

X Toolkit Intrinsics - C Language Interface

Bitmap Distribution Format 2.1
This standard has been adopted for Federal Government use.

Details concerning its use within the Federal Government are contained in Federal Information Processing Standards Publication 158-1, The User Interface Component of the Applications Portability Profile. For a complete list of the publications available in the Federal Information Processing Standards series, write to the Standards Processing Coordinator (ADP), National Institute of Standards and Technology, Gaithersburg, MD 20899.
X Window System Protocol
MIT X Consortium Standard
X Version 11, Release 5

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

The primary contributors 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
# Table of Contents

Acknowledgments .................................................................................. iii
1. Protocol Formats .................................................................................. 1
2. Syntactic Conventions ......................................................................... 1
3. Common Types .................................................................................. 2
4. Errors .................................................................................................. 4
5. Keyboards .......................................................................................... 5
6. Pointers .............................................................................................. 6
7. Predefined Atoms ............................................................................... 6
8. Connection Setup ............................................................................... 7
9. Requests ............................................................................................ 11
10. Connection Close ............................................................................. 63
11. Events .............................................................................................. 64
12. Flow Control and Concurrency .......................................................... 74
Appendix A – KEYSYM Encoding ............................................................. 74
Appendix B – Protocol Encoding .............................................................. 93
Glossary ................................................................................................. 138
Index .................................................................................................. 146
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 4), 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 9 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 11 are described in the following format:

  EventName
  value1 : type1
  ...
  valueN: typeN

Description.

3. Common Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISTofFOO</td>
<td>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.</td>
</tr>
<tr>
<td>BITMASK</td>
<td>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.</td>
</tr>
<tr>
<td>LISTofVALUE</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>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.</td>
</tr>
<tr>
<td>WINDOW</td>
<td>32-bit value (top three bits guaranteed to be zero)</td>
</tr>
<tr>
<td>PIXMAP</td>
<td>32-bit value (top three bits guaranteed to be zero)</td>
</tr>
<tr>
<td>CURSOR</td>
<td>32-bit value (top three bits guaranteed to be zero)</td>
</tr>
<tr>
<td>FONT</td>
<td>32-bit value (top three bits guaranteed to be zero)</td>
</tr>
<tr>
<td>GCONTEXT</td>
<td>32-bit value (top three bits guaranteed to be zero)</td>
</tr>
<tr>
<td>COLORMAP</td>
<td>32-bit value (top three bits guaranteed to be zero)</td>
</tr>
<tr>
<td>DRAWABLE</td>
<td>WINDOW or PIXMAP</td>
</tr>
</tbody>
</table>
FONTABLE
ATOM
VISUALID
VALUE
BYTE
INT8
INT16
INT32
CARD8
CARD16
CARD32
TIMESTAMP
BITGRAVITY
WINGRAVITY
BOOL
EVENT
POINTEREVENT
DEVICEEVENT
KEYSYM
KEYCODE
BUTTON
KEYMASK
BUTMASK
KEYBUTFMASK
STRING8
STRING16
CHAR2B
POINT
RECTANGLE
ARC
HOST

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:

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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, or to free an entry in a colormap that was created with all entries writable.  
             | 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. |
| 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 COLOMAP argument does not name a defined COLOMAP. |
| 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.  
<pre><code>         | A value for a FONTABLE argument does not name a defined FONT or a defined GCONTEXT. |
</code></pre>
<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GContext</td>
<td>A value for a GCONTEXT argument does not name a defined GCONTEXT.</td>
</tr>
<tr>
<td>IDChoice</td>
<td>The value chosen for a resource identifier either is not included in the range assigned to the client or is already in use.</td>
</tr>
<tr>
<td>Implementation</td>
<td>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.</td>
</tr>
<tr>
<td>Length</td>
<td>The length of a request is shorter or longer than that required to minimally contain the arguments.</td>
</tr>
<tr>
<td>Match</td>
<td>The length of a request exceeds the maximum length accepted by the server.</td>
</tr>
<tr>
<td>Name</td>
<td>An InputOnly window is used as a DRAWABLE.</td>
</tr>
<tr>
<td>Pixmap</td>
<td>In a graphics request, the GCONTEXT argument does not have the same root and depth as the destination DRAWABLE argument.</td>
</tr>
<tr>
<td>Request</td>
<td>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.</td>
</tr>
<tr>
<td>Value</td>
<td>A font or color of the specified name does not exist.</td>
</tr>
<tr>
<td>Window</td>
<td>A value for a WINDOW argument does not name a defined WINDOW.</td>
</tr>
</tbody>
</table>

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, Cyrillic, 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 Void-Symbol 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 the 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 defined. The modifier state determines which group to use. 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, the modifier state determines which KEYSYM to use. 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.

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

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.

Connection Initiation

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."

**Server Response**

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:

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

Server Information
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, GC context, and COLOR- MAP 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, visual ids, 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.

Screen Information

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.

Backings-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.

Visual Information
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:

\[ \text{hw-intensity} = \frac{\text{protocol-intensity}}{(65536 / \text{total-hw-intensities})} \]

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

\[
\begin{align*}
\text{wid}, \text{parent} &: \text{WINDOW} \\
\text{class}: \{ \text{InputOutput}, \text{InputOnly}, \text{CopyFromParent} \} \\
\text{depth}: \text{CARD8} \\
\text{visual}: \text{VISUALID or CopyFromParent} \\
x, y &: \text{INT16} \\
\text{width}, \text{height}, \text{border-width} &: \text{CARD16} \\
\text{value-mask}: \text{BITMASK} \\
\text{value-list}: \text{LISTofVALUE}
\end{align*}
\]

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

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 coordinate system has the X axis horizontal and the Y axis vertical, with the origin \([0, 0]\) at the upper left. Coordinates are integral, in terms of pixels, and coincide with pixel centers. Each window and pixmap has its own coordinate system. For a window, the origin is inside the border at the inside upper left.

The x and y coordinates for the window 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 window is placed on top in the stacking order with respect to siblings.

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

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

The default values when attributes are not explicitly initialized are:

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

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.
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 and root 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
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*: Match, Value, Window

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 10.

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

**ReparentWindow**

*window*, *parent*: WINDOW  
*x*, *y*: INT16  
*Errors*: Match, Window

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 new parent is InputOnly and the window is not.
- 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: Match, Value, Window

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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Deltas</th>
</tr>
</thead>
<tbody>
<tr>
<td>NorthWest</td>
<td>([0, 0])</td>
</tr>
<tr>
<td>North</td>
<td>([W/2, 0])</td>
</tr>
<tr>
<td>NorthEast</td>
<td>([W, 0])</td>
</tr>
<tr>
<td>West</td>
<td>([0, H/2])</td>
</tr>
<tr>
<td>Center</td>
<td>([W/2, H/2])</td>
</tr>
<tr>
<td>East</td>
<td>([W, H/2])</td>
</tr>
<tr>
<td>SouthWest</td>
<td>([0, H])</td>
</tr>
<tr>
<td>South</td>
<td>([W/2, H])</td>
</tr>
<tr>
<td>SouthEast</td>
<td>([W, H])</td>
</tr>
</tbody>
</table>
When a window with one of these bit-gravities is resized, the corresponding pair defines the change in position of each pixel in the window. When a window with one of these win-gravities has its parent window resized, the corresponding pair defines the change in position of the window within the parent. 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**: Value, Window

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**: Alloc, Value

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 remain defined until server reset (see section 10).

GetAtomName

\[
\begin{align*}
\text{atom: ATOM} \\
\Rightarrow \\
\text{name: STRING8} \\
\text{Errors: Atom}
\end{align*}
\]

This request returns the name for the given atom.

ChangeProperty

\[
\begin{align*}
\text{window: WINDOW} \\
\text{property, type: ATOM} \\
\text{format: \{8, 16, 32\}} \\
\text{mode: \{Replace, Prepend, Append\}} \\
\text{data: LISTofINT8 or LISTofINT16 or LISTofINT32} \\
\text{Errors: Alloc, Atom, Match, Value, Window}
\end{align*}
\]

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 10).

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

DeleteProperty

\[
\begin{align*}
\text{window: WINDOW} \\
\text{property: ATOM} \\
\text{Errors: Atom, Window}
\end{align*}
\]

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

\[
\begin{align*}
\text{window: WINDOW} \\
\text{property: ATOM} \\
\text{type: ATOM or AnyPropertyType} \\
\text{long-offset, long-length: CARD32} \\
\text{delete: BOOL} \\
\Rightarrow \\
\text{type: ATOM or None} \\
\text{format: \{0, 8, 16, 32\}} \\
\text{bytes-after: CARD32}
\end{align*}
\]

21
value: LISTofINT8 or LISTofINT16 or LISTofINT32

Errors: Atom, Value, Window

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:

\[
N = \text{actual length of the stored property in bytes} \\
I = 4 \times \text{long-offset} \\
T = N - I \\
L = \text{MINIMUM}(T, 4 \times \text{long-length}) \\
A = N - (I + L)
\]

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: Atom, Match, Window

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 \((I + \text{delta}) \mod N\), 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: Value, Window

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

\[
\begin{align*}
grab-window: & \text{ WINDOW} \\
owner-events: & \text{ BOOL} \\
event-mask: & \text{ SETofPOINTEREVENT} \\
pointer-mode, keyboard-mode: & \{\text{Synchronous, Asynchronous}\} \\
confine-to: & \text{ WINDOW or None} \\
cursor: & \text{ CURSOR or None} \\
time: & \text{ TIMESTAMP or CurrentTime} \\
\end{align*}
\]

=>

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

Errors: Cursor, Value, Window

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

\[ time: \text{TIMESTAMP or CurrentTime} \]

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

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

Errors: Access, Cursor, Value, Window

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: Value, Window

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: Value, Window

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 \texttt{False}, all generated key events are reported with respect to grab-window.
If owner-events is \texttt{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 \texttt{KeyPress} and \texttt{KeyRelease} events are always reported, independent of any
event selection made by the client.

If keyboard-mode is \texttt{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 \texttt{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 \texttt{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 \texttt{Asynchronous}, pointer event processing is unaffected by activation of the
grab. If pointer-mode is \texttt{Synchronous}, the state of the pointer (as seen by means of the proto¬
col) appears to freeze. No further pointer events are generated by the server until the grabbing
client issues a releasing \texttt{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 pro¬
cessing.

This request generates \texttt{FocusIn} and \texttt{FocusOut} events.

The request fails with status \texttt{AlreadyGrabbed} if the keyboard is actively grabbed by some
other client. The request fails with status \texttt{Frozen} if the keyboard is frozen by an active grab
of another client. The request fails with status \texttt{NotViewable} if grab-window is not viewable.
The request fails with status \texttt{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 \texttt{CurrentTime} replaced by the current server time.

\texttt{UngrabKeyboard}

\texttt{time: TIMESTAMP or CurrentTime}

This request releases the keyboard if this client has it actively grabbed (as a result of either
\texttt{GrabKeyboard} or \texttt{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 \texttt{FocusIn} and \texttt{FocusOut} events.

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

\texttt{GrabKey}

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

Errors: Access, Value, Window

This request establishes a passive grab on the keyboard. In the future, the keyboard is actively
grabbed as described in \texttt{GrabKeyboard}, the last-keyboard-grab time is set to the time at
which the key was pressed (as transmitted in the \texttt{KeyPress} event), and the \texttt{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.

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.

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, but the keyboard need not be 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
\[
\text{window: WINDOW}
\]
\[
\Rightarrow
\]
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
\[
\text{start, stop: TIMESTAMP or CurrentTime}
\]
\[
\text{window: WINDOW}
\]
\[
\Rightarrow
\]
events: LISTofTIMECOORD
where:

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

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
\[
\text{src-window, dst-window: WINDOW}
\]
\[
\text{src-x, src-y: INT16}
\]
\[
\Rightarrow
\]
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

\[
\text{src-window: WINDOW or None} \\
\text{dst-window: WINDOW or None} \\
\text{src-x, src-y: INT16} \\
\text{src-width, src-height: CARD16} \\
\text{dst-x, dst-y: INT16}
\]

Errors: Window

If dst-window is None, this request moves the pointer by offsets \([\text{dst-x, dst-y}]\) relative to the current position of the pointer. If dst-window is a window, this request moves the pointer to \([\text{dst-x, dst-y}]\) 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

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

Errors: Match, Value, Window

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 \text{Current-Time} 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

font: FONT
name: STRING8

Errors: Alloc, IDChoice, Name

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
cchar-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]
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:

\[
\begin{align*}
\text{byte1} & = N/D + \text{min-byte1} \\
\text{byte2} & = N \mod D + \text{min-char-or-byte2}
\end{align*}
\]

where:

\[
\begin{align*}
D & = \text{max-char-or-byte2} - \text{min-char-or-byte2} + 1 \\
/ & = \text{integer division} \\
\mod & = \text{integer modulus}
\end{align*}
\]

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:

\[
L = D \times (\text{max-byte1} - \text{min-byte1} + 1)
\]

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:

\[
[x + \text{min-bounds.left-side-bearing}, y - \text{max-bounds.ascent}]
\]

with a width of:

\[
\text{max-bounds.right-side-bearing} - \text{min-bounds.left-side-bearing}
\]

and a height of:

\[
\text{max-bounds.ascent} + \text{max-bounds.descent}
\]
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:

\([x + \text{left-side-bearing}, y - \text{ascent}]\)

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:

\([x + \text{character-width}, y]\)

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

\(\text{font}: \text{FONTABLE} \quad \text{string}: \text{STRING16} \Rightarrow \)

\(\text{draw-direction}: \{ \text{LeftToRight, RightToLeft} \}
\text{font-ascent}: \text{INT16}
\text{font-descent}: \text{INT16}
\text{overall-ascent}: \text{INT16}
\text{overall-descent}: \text{INT16}
\text{overall-width}: \text{INT32}
\text{overall-left}: \text{INT32}
\text{overall-right}: \text{INT32}
\) 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 \texttt{CHAR2B} as a 16-bit number that has been transmitted most-significant byte first (that is, byte1 of the \texttt{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**

```plaintext
pattern: STRING8
max-names: CARD16
=>
  names: LISTofSTRING8
```

This request returns a list of available font names (as controlled by the font search path; see \texttt{SetFontPath} request) that match the pattern. At most, \texttt{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 \texttt{\?} character (octal value 77) will match any single character, and the \texttt{\*\*} character (octal value 52) will match any number of characters. The returned names are in lowercase.

**ListFontsWithInfo**

```plaintext
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 \texttt{ListFonts}, but it also returns information about each font. The information returned for each font is identical to what \texttt{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**

```plaintext
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**

```plaintext
=>
path: LISTofSTRING8
```

This request returns the current search path for fonts.

**CreatePixmap**

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

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

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**

```plaintext
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**

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

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

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:

<table>
<thead>
<tr>
<th>Component</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>{Clear, And, AndReverse, Copy, AndInverted, NoOp, Xor, Or, Nor, Equiv, Invert, OrReverse, CopyInverted, OrInverted, Nand, Set}</td>
</tr>
<tr>
<td>plane-mask</td>
<td>CARD32</td>
</tr>
<tr>
<td>foreground</td>
<td>CARD32</td>
</tr>
<tr>
<td>background</td>
<td>CARD32</td>
</tr>
<tr>
<td>line-width</td>
<td>CARD16</td>
</tr>
</tbody>
</table>
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:

\[
((\text{src FUNC dst}) \text{ AND plane-mask}) \text{ OR (dst AND (NOT plane-mask))}
\]

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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear</td>
<td>0</td>
</tr>
<tr>
<td>And</td>
<td>src AND dst</td>
</tr>
<tr>
<td>AndReverse</td>
<td>src AND (NOT dst)</td>
</tr>
<tr>
<td>Copy</td>
<td>src</td>
</tr>
<tr>
<td>AndInverted</td>
<td>(NOT src) AND dst</td>
</tr>
<tr>
<td>NoOp</td>
<td>dst</td>
</tr>
<tr>
<td>Xor</td>
<td>src XOR dst</td>
</tr>
<tr>
<td>Or</td>
<td>src OR dst</td>
</tr>
<tr>
<td>Nor</td>
<td>(NOT src) AND (NOT dst)</td>
</tr>
<tr>
<td>Equiv</td>
<td>(NOT src) XOR dst</td>
</tr>
<tr>
<td>Invert</td>
<td>NOT dst</td>
</tr>
<tr>
<td>OrReverse</td>
<td>src OR (NOT dst)</td>
</tr>
<tr>
<td>CopyInverted</td>
<td>NOT src</td>
</tr>
<tr>
<td>OrInverted</td>
<td>(NOT src) OR dst</td>
</tr>
<tr>
<td>Nand</td>
<td>(NOT src) OR (NOT dst)</td>
</tr>
<tr>
<td>Set</td>
<td>1</td>
</tr>
</tbody>
</table>

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 \([x_1, y_1], [x_2, y_2]\) and width \(w\) is a rectangle with vertices at the following real coordinates:

\[
\begin{align*}
[x_1-(w*\text{sn}/2), y_1+(w*\text{cs}/2)],
[x_1+(w*\text{sn}/2), y_1-(w*\text{cs}/2)],
[x_2-(w*\text{sn}/2), y_2+(w*\text{cs}/2)],
[x_2+(w*\text{sn}/2), y_2-(w*\text{cs}/2)]
\end{align*}
\]

The \text{sn} is the sine of the angle of the line and \text{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 \([x_1, y_1]\) to \([x_2, y_2]\) and another line is drawn unclipped from \([x_1+dx, y_1+dy]\) to \([x_2+dx, y_2+dy]\), 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 \([x_1, y_1]\) to \([x_2, y_2]\) always draws the same pixels as a wide line drawn from \([x_2, y_2]\) to \([x_1, y_1]\), 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.
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 \((x_1=x_2, y_1=y_2)\), 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 \((x_1=x_2, y_1=y_2)\), 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 represents an infinite 2D plane, with the tile/stipple replicated in all dimensions. When that plane is superimposed on the drawable for use in a graphics operation, the upper left corner of some instance of the tile/stipple is at the coordinates within the drawable specified by the tile/stipple origin. 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**

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.

**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:
<table>
<thead>
<tr>
<th>Component</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>Copy</td>
</tr>
<tr>
<td>plane-mask</td>
<td>all ones</td>
</tr>
<tr>
<td>foreground</td>
<td>0</td>
</tr>
<tr>
<td>background</td>
<td>1</td>
</tr>
<tr>
<td>line-width</td>
<td>0</td>
</tr>
<tr>
<td>line-style</td>
<td>Solid</td>
</tr>
<tr>
<td>cap-style</td>
<td>Butt</td>
</tr>
<tr>
<td>join-style</td>
<td>Miter</td>
</tr>
<tr>
<td>fill-style</td>
<td>Solid</td>
</tr>
<tr>
<td>fill-rule</td>
<td>EvenOdd</td>
</tr>
<tr>
<td>arc-mode</td>
<td>PieSlice</td>
</tr>
<tr>
<td>tile</td>
<td>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)</td>
</tr>
<tr>
<td>stipple</td>
<td>Pixmap of unspecified size filled with ones</td>
</tr>
<tr>
<td>tile-stipple-x-origin</td>
<td>0</td>
</tr>
<tr>
<td>tile-stipple-y-origin</td>
<td>0</td>
</tr>
<tr>
<td>font</td>
<td>&lt;server-dependent-font&gt;</td>
</tr>
<tr>
<td>subwindow-mode</td>
<td>ClipByChildren</td>
</tr>
<tr>
<td>graphics-exposures</td>
<td>True</td>
</tr>
<tr>
<td>clip-x-origin</td>
<td>0</td>
</tr>
<tr>
<td>clip-y-origin</td>
<td>0</td>
</tr>
<tr>
<td>clip-mask</td>
<td>None</td>
</tr>
<tr>
<td>dash-offset</td>
<td>0</td>
</tr>
<tr>
<td>dashes</td>
<td>4 (that is, the list [4, 4])</td>
</tr>
</tbody>
</table>

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

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: Alloc, GContext, Match, Value

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: Alloc, GContext, Value

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 135 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: Alloc, GContext, Match, Value

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: Match, Value, Window
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
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, Match, Value

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, Match, Value

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, 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

**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 \([x_1, y_1, w_1, h_1, a_1, a_2]\), the origin of the major and minor axes is at \([x_1+(w_1/2), y_1+(h_1/2)]\), and the infinitely thin path describing the entire circle/ellipse intersects the horizontal axis at \([x_1, y_1+(h_1/2)]\) and \([x_1+w_1, y_1+(h_1/2)]\) and intersects the vertical axis at \([x_1+(w_1/2), y_1]\) and \([x_1+(w_1/2), y_1+h_1]\). 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 \([x_1, y_1, w_1, h_1, a_1, a_2]\), 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:

\[
\text{skewed-angle} = \text{atan}(\text{tan(normal-angle)} * \frac{w_1}{h_1}) + \text{adjust}
\]

The skewed-angle and normal-angle are expressed in radians (rather than in degrees scaled by 64) in the range \([0, 2\pi]\). The \(\text{atan}\) returns a value in the range \([-\pi/2, \pi/2]\). The \(\text{adjust}\) 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-
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:

\[
[x,y] \ [x+\text{width},y] \ [x+\text{width},y+\text{height}] \ [x,y+\text{height}]
\]

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, Match, Value

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**

```plaintext
drawable: DRAWABLE
gc: GCONTEXT
x, y: INT16
items: LISTofTEXTITEM8
where:

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

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

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 bytel 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**

```plaintext
drawable: DRAWABLE
gc: GCONTEXT
x, y: INT16
items: LISTofTEXTITEM16
where:

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

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

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, byte 1 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:

\[ [x, y - \text{font-ascent}] \]

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, byte 1 of the CHAR2B is taken as the most-significant byte).

**CreateColormap**

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

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

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: Alloc, Colormap, IDChoice

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 minimum-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 \texttt{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 \texttt{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).

\textbf{UninstallColormap}

\begin{verbatim}
cmap: COLORMAP

Errors: Colormap
\end{verbatim}

If \texttt{cmap} is on the required list for its screen (see \texttt{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 \texttt{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.

\textbf{ListInstalledColormaps}

\begin{verbatim}
window: WINDOW
=>
cmaps: LISTofCOLORMAP

Errors: Window
\end{verbatim}

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 \texttt{InstallColormap} request).

\textbf{AllocColor}

\begin{verbatim}
cmap: COLORMAP
red, green, blue: CARD16

=>

pixel: CARD32
red, green, blue: CARD16

Errors: Alloc, Colormap
\end{verbatim}

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.

\textbf{AllocNamedColor}
cmap:: COLORMAP
ame:: STRING8
=>
  pixel:: CARD32
  exact-red, exact-green, exact-blue:: CARD16
  visual-red, visual-green, visual-blue:: CARD16
Errors:: Alloc, Colormap, Name

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:: Alloc, Colormap, Value

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, \(C \times 2^P\) 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:: Alloc, Colormap, Value

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, \(C \times 2^R \times 2^G \times 2^B\) distinct pixels can be produced; all of these are allocated writable by the request. The initial RGB values of the allocated entries are undefined. In the colormap, there are only \(C \times 2^R\) independent red entries, \(C \times 2^G\) independent green entries, and \(C \times 2^B\) 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

\textit{cmap}: \texttt{COLORMAP}
\textit{pixels}: \texttt{LISTofCARD32}
\textit{plane-mask}: \texttt{CARD32}

\textbf{Errors}: Access, Colormap, 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 \texttt{AllocColor}, \texttt{AllocNamedColor}, \texttt{AllocColorCells}, and \texttt{AllocColorPlanes}). Note that freeing an individual pixel obtained from \texttt{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 \texttt{cmap} are freed, even if one or more pixels produce an error. A \texttt{Value} error is generated if a specified pixel is not a valid index into \texttt{cmap}. An \texttt{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) or if the colormap was created with all entries writable (using an alloc value of \texttt{All} in \texttt{CreateColormap}). If more than one pixel is in error, it is arbitrary as to which pixel is reported.

StoreColors

\textit{cmap}: \texttt{COLORMAP}
\textit{items}: \texttt{LISTofCOLORITEM}

where:

\texttt{COLORITEM}: 
\begin{itemize}
  \item \texttt{[pixel: CARD32 do-red, do-green, do-blue: BOOL red, green, blue: CARD16]}
\end{itemize}

\textbf{Errors}: Access, Colormap, 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 \texttt{cmap} (by any client) are changed, even if one or more pixels produce an error. A \texttt{Value} error is generated if a specified pixel is not a valid index into \texttt{cmap}, and an \texttt{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

\textit{cmap}: \texttt{COLORMAP}
\textit{pixel}: \texttt{CARD32}
\textit{name}: \texttt{STRING8}
\textit{do-red, do-green, do-blue}: \texttt{BOOL}

\textbf{Errors}: Access, Colormap, Name, Value

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

QueryColors
 colormap: 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
 colormap: 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: Alloc, IDChoice, Match, Pixmap

 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

\[\text{cid: CURSOR} \]
\[\text{source-font: FONT} \]
\[\text{mask-font: FONT or None} \]
\[\text{source-char, mask-char: CARD16} \]
\[\text{fore-red, fore-green, fore-blue: CARD16} \]
\[\text{back-red, back-green, back-blue: CARD16} \]

Errors: Alloc, Font, IDChoice, Value

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

\[\text{cursor: CURSOR} \]

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

\[\text{cursor: CURSOR} \]
\[\text{fore-red, fore-green, fore-blue: CARD16} \]
\[\text{back-red, back-green, back-blue: CARD16} \]

Errors: Cursor

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

Query Best Size

\[\text{class: \{Cursor, Tile, Stipple\}} \]
\[\text{drawable: DRAWABLE} \]
\[\text{width, height: CARD16} \]

Errors: Drawable, Match, Value

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

\[name: \text{STRING8}\]
\[
>=
\]
\[
present: \text{BOOL}
\]
\[
major-opcode: \text{CARD8}
\]
\[
first-event: \text{CARD8}
\]
\[
first-error: \text{CARD8}
\]

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

\[
>=
\]
\[
names: \text{LISTofSTRING8}
\]

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

SetModifierMapping

\[keycodes-per-modifier: \text{CARD8}\]
\[keycodes: \text{LISTofKEYCODE}\]
\[
>=
\]
\[
status: \{\text{Success}, \text{Busy}, \text{Failed}\}
\]

Errors: Alloc, Value

This request specifies the keycodes (if any) of the keys to be used as modifiers. The number of keycodes in the list must be \(8 \times \text{keycodes-per-modifier}\) (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 is 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: Alloc, Value

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:

\[
first-keycode + \left( \frac{\text{keysyms-length}}{\text{keysyms-per-keycode}} \right) - 1
\]

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:

\[
(K - \text{first-keycode}) * \text{keysyms-per-keycode} + N
\]

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

This request generates a MappingNotify event.

There is no requirement that the server interpret this mapping; it is merely stored for reading and writing by clients (see section 5).

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:

\[
first-keycode + \text{count} - 1
\]

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:

\[(K - \text{first-keycode}) \times \text{keysyms-per-keycode} + N\]

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:

<table>
<thead>
<tr>
<th>Control</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>key-click-percent</td>
<td>INT8</td>
</tr>
<tr>
<td>bell-percent</td>
<td>INT8</td>
</tr>
<tr>
<td>bell-pitch</td>
<td>INT16</td>
</tr>
<tr>
<td>bell-duration</td>
<td>INT16</td>
</tr>
<tr>
<td>led</td>
<td>CARD8</td>
</tr>
<tr>
<td>led-mode</td>
<td>[On, Off]</td>
</tr>
<tr>
<td>key</td>
<td>KEYCODE</td>
</tr>
<tr>
<td>auto-repeat-mode</td>
<td>[On, Off, Default]</td>
</tr>
</tbody>
</table>

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.

59
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:

\[ \text{base} - \left( \frac{\text{base} \times \text{percent}}{100} \right) + \text{percent} \]

When percent is negative, it is:

\[ \text{base} + \left( \frac{\text{base} \times \text{percent}}{100} \right) \]

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, screensaver 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 made. Examples of ways to change the screen include scrambling the color map periodically, moving an icon image about the screen periodically, or tiling the screen with the root window background tile, randomly reorigined periodically.

GetScreenSaver
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\}

*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*: Access, Value

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 10.

**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 10). 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:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>event-window</td>
<td>Event window</td>
</tr>
<tr>
<td>event-mask</td>
<td>Client’s selected pointer events on the event window</td>
</tr>
<tr>
<td>pointer-mode and keyboard-mode</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>owner-events</td>
<td>True if the client has OwnerGrabButton selected on the event window, otherwise False</td>
</tr>
<tr>
<td>confine-to</td>
<td>None</td>
</tr>
<tr>
<td>cursor</td>
<td>None</td>
</tr>
</tbody>
</table>

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
cchild: 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:

<table>
<thead>
<tr>
<th>Event Component Type</th>
<th>Detail Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyPress, KeyRelease</td>
<td>KEYCODE</td>
</tr>
<tr>
<td>ButtonPress, ButtonRelease</td>
<td>BUTTON</td>
</tr>
<tr>
<td>MotionNotify</td>
<td>{Normal, Hint}</td>
</tr>
</tbody>
</table>

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

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

\text{event: WINDOW}

\text{mode: \{Normal, WhileGrabbed, Grab, Ungrab\}}

\text{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:

- \text{FocusOut with detail Ancestor is generated on A.}
- \text{FocusOut with detail Virtual is generated on each window between A and B exclusive (in order).}
- \text{FocusIn with detail Inferior is generated on B.}
- \text{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:

- \text{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).}
- \text{FocusOut with detail Inferior is generated on A.}
- \text{FocusIn with detail Virtual is generated on each window between A and B exclusive (in order).}
- \text{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:

- \text{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).}
- \text{FocusOut with detail Nonlinear is generated on A.}
- \text{FocusOut with detail NonlinearVirtual is generated on each window between A and C exclusive (in order).}
- \text{FocusIn with detail NonlinearVirtual is generated on each window between C and B exclusive (in order).}
- \text{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.

KeypmapNotify

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 Visibility 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

\[
\begin{align*}
\text{drawable: DRA WABLE} \\
\text{major-opcode: CARD8} \\
\text{minor-opcode: CARD16}
\end{align*}
\]

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

\[
\begin{align*}
\text{window: WINDOW} \\
\text{state: \{Unobscured, PartiallyObscured, FullyObscured\}}
\end{align*}
\]

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

\[
\begin{align*}
\text{parent, window: WINDOW} \\
x, y: INT16 \\
width, height, border-width: CARD16 \\
\text{override-redirect: BOOL}
\end{align*}
\]

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

\[
\begin{align*}
\text{event, window: WINDOW}
\end{align*}
\]

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

\[ \text{event, window: WINDOW} \]
\[ \text{from-configure: BOOL} \]

This event is reported to clients selecting \text{StructureNotify} on the window and to clients selecting \text{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 \text{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

\[ \text{event, window: WINDOW} \]
\[ \text{override-redirect: BOOL} \]

This event is reported to clients selecting \text{StructureNotify} on the window and to clients selecting \text{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

\[ \text{parent, window: WINDOW} \]

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

ReparentNotify

\[ \text{event, window, parent: WINDOW} \]
\[ x, y: \text{INT16} \]
\[ \text{override-redirect: BOOL} \]

This event is reported to clients selecting \text{SubstructureNotify} on either the old or the new parent and to clients selecting \text{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

\[ \text{event, window: WINDOW} \]
\[ x, y: \text{INT16} \]
\[ width, height, border-width: \text{CARD16} \]
\[ above-sibling: \text{WINDOW or None} \]
\[ \text{override-redirect: BOOL} \]

This event is reported to clients selecting \text{StructureNotify} on the window and to clients selecting \text{SubstructureNotify} on the parent. It is generated when a \text{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 \text{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

\[ \text{event, window: WINDOW} \]
\[ x, y: \text{INT16} \]

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

\[ \text{window: WINDOW} \]
\[ \text{width, height: CARD16} \]

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

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

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

\[ \text{event, window: WINDOW} \]
\[ \text{place: \{ Top, Bottom \}} \]

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

\[ \text{parent, window: WINDOW} \]
\[ \text{place: \{ Top, Bottom \}} \]

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.

Appendix A
KEYSYM Encoding

For convenience, KEYSYM values are viewed as split into four bytes:
- Byte 1 (for the purposes of this encoding) is the most-significant 5 bits (because of the 29-bit effective values)
- Byte 2 is the next most-significant 8 bits
- Byte 3 is the next most-significant 8 bits
- Byte 4 is the least-significant 8 bits

There are two special KEYSYM values: NoSymbol and VoidSymbol. They are used to indicate the absence of symbols (see section 5).

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NoSymbol</td>
</tr>
<tr>
<td>0</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>VoidSymbol</td>
</tr>
</tbody>
</table>

All other standard KEYSYM values have zero values for bytes 1 and 2. Byte 3 indicates a character code set, and byte 4 indicates a particular character within that set.

<table>
<thead>
<tr>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Latin 1</td>
</tr>
<tr>
<td>1</td>
<td>Latin 2</td>
</tr>
<tr>
<td>2</td>
<td>Latin 3</td>
</tr>
<tr>
<td>3</td>
<td>Latin 4</td>
</tr>
<tr>
<td>4</td>
<td>Kana</td>
</tr>
<tr>
<td>5</td>
<td>Arabic</td>
</tr>
<tr>
<td>6</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>7</td>
<td>Greek</td>
</tr>
<tr>
<td>8</td>
<td>Technical</td>
</tr>
<tr>
<td>9</td>
<td>Special</td>
</tr>
<tr>
<td>10</td>
<td>Publishing</td>
</tr>
<tr>
<td>11</td>
<td>APL</td>
</tr>
<tr>
<td>12</td>
<td>Hebrew</td>
</tr>
<tr>
<td>255</td>
<td>Keyboard</td>
</tr>
</tbody>
</table>

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

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

To the best of our knowledge, the Latin, Kana, Arabic, Cyrillic, Greek, APL, and Hebrew sets are from the appropriate ISO and/or ECMA international standards. There are no Technical, Special, or Publishing international standards, so these sets are based on Digital Equipment Corporation standards.

The ordering between the sets (byte 3) is essentially arbitrary. National and international standards bodies were commencing deliberations regarding international 2-byte and 4-byte character sets at the time these keysyms were developed, but we did not know of any proposed layouts.
The order may be arbitrary, but it is important in dealing with duplicate coding. As far as possible, keysym values (byte 4) follow the character set encoding standards, except for the Greek and Cyrillic keysyms which are based on early draft standards. In the Latin-1 to Latin-4 sets, all duplicate glyphs occupy the same code position. However, duplicates between Greek and Technical do not occupy the same code position. Applications that wish to use the Latin-2, Latin-3, Latin-4, Greek, Cyrillic, or Technical sets may find it convenient to use arrays to transform the keysyms.

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

<table>
<thead>
<tr>
<th>US name</th>
<th>European name</th>
<th>code position in Latin-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section sign</td>
<td>Paragraph sign</td>
<td>10/07</td>
</tr>
<tr>
<td>Paragraph sign</td>
<td>Pilcrow sign</td>
<td>11/06</td>
</tr>
</tbody>
</table>

We have adopted the US names (by accident rather than by design).

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

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

- The Engraving approach
- The Common approach

The Engraving approach is to create a keysym for every unique key engraving. This is effectively taking the union of all key engravings on all keyboards. For example, some keyboards label function keys across the top as F1 through Fn, and others label them as PF1 through PFn. These would be different keys under the Engraving approach. Likewise, Lock would differ from Shift Lock, which is different from the up-arrow symbol that has the effect of changing lowercase to uppercase. There are lots of other aliases such as Del, DEL, Delete, Remove, and so forth. The Engraving approach makes it easy to decide if a new entry should be added to the keysym set: if it does not exactly match an existing one, then a new one is created. One estimate is that there would be on the order of 300–500 Keyboard keysyms using this approach, without counting foreign translations and variations.

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

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

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

\[ \text{byte3} \times 256 + \text{byte4} \]

<table>
<thead>
<tr>
<th>Byte3</th>
<th>Byte4</th>
<th>Code Pos</th>
<th>Name</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>032</td>
<td>02/00</td>
<td>SPACE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>033</td>
<td>02/01</td>
<td>EXCLAMATION POINT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>034</td>
<td>02/02</td>
<td>QUOTATION MARK</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>035</td>
<td>02/03</td>
<td>NUMBER SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>036</td>
<td>02/04</td>
<td>DOLLAR SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>037</td>
<td>02/05</td>
<td>PERCENT SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>038</td>
<td>02/06</td>
<td>AMPERSAND</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>039</td>
<td>02/07</td>
<td>POINT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>040</td>
<td>02/08</td>
<td>LEFT PARENTHESES</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>041</td>
<td>02/09</td>
<td>RIGHT PARENTHESES</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>042</td>
<td>02/10</td>
<td>ASTERISK</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>043</td>
<td>02/11</td>
<td>PLUS SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>044</td>
<td>02/12</td>
<td>COMMA</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>045</td>
<td>02/13</td>
<td>MINUS SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>046</td>
<td>02/14</td>
<td>FULL STOP</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>047</td>
<td>02/15</td>
<td>SOLIDUS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>048</td>
<td>03/00</td>
<td>DIGIT ZERO</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>049</td>
<td>03/01</td>
<td>DIGIT ONE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>050</td>
<td>03/02</td>
<td>DIGIT TWO</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>051</td>
<td>03/03</td>
<td>DIGIT THREE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>052</td>
<td>03/04</td>
<td>DIGIT FOUR</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>053</td>
<td>03/05</td>
<td>DIGIT FIVE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>054</td>
<td>03/06</td>
<td>DIGIT SIX</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>055</td>
<td>03/07</td>
<td>DIGIT SEVEN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>056</td>
<td>03/08</td>
<td>DIGIT EIGHT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>057</td>
<td>03/09</td>
<td>DIGIT NINE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>058</td>
<td>03/10</td>
<td>COLON</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>059</td>
<td>03/11</td>
<td>SEMICOLON</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>060</td>
<td>03/12</td>
<td>LESS THAN SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>061</td>
<td>03/13</td>
<td>EQUALS SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>062</td>
<td>03/14</td>
<td>GREATER THAN SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>063</td>
<td>03/15</td>
<td>QUESTION MARK</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>064</td>
<td>04/00</td>
<td>COMMERCIAL AT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>065</td>
<td>04/01</td>
<td>LATIN CAPITAL LETTER A</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>066</td>
<td>04/02</td>
<td>LATIN CAPITAL LETTER B</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>067</td>
<td>04/03</td>
<td>LATIN CAPITAL LETTER C</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>068</td>
<td>04/04</td>
<td>LATIN CAPITAL LETTER D</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>069</td>
<td>04/05</td>
<td>LATIN CAPITAL LETTER E</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>070</td>
<td>04/06</td>
<td>LATIN CAPITAL LETTER F</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>071</td>
<td>04/07</td>
<td>LATIN CAPITAL LETTER G</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>072</td>
<td>04/08</td>
<td>LATIN CAPITAL LETTER H</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>073</td>
<td>04/09</td>
<td>LATIN CAPITAL LETTER I</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>074</td>
<td>04/10</td>
<td>LATIN CAPITAL LETTER J</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>075</td>
<td>04/11</td>
<td>LATIN CAPITAL LETTER K</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>076</td>
<td>04/12</td>
<td>LATIN CAPITAL LETTER L</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>077</td>
<td>04/13</td>
<td>LATIN CAPITAL LETTER M</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>078</td>
<td>04/14</td>
<td>LATIN CAPITAL LETTER N</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>079</td>
<td>04/15</td>
<td>LATIN CAPITAL LETTER O</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>080</td>
<td>05/00</td>
<td>LATIN CAPITAL LETTER P</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>081</td>
<td>05/01</td>
<td>LATIN CAPITAL LETTER Q</td>
<td>Latin-1</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Byte 4</td>
<td>Code Pos</td>
<td>Name</td>
<td>Set</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>----------</td>
<td>-----------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>000</td>
<td>082</td>
<td>05/02</td>
<td>LATIN CAPITAL LETTER R</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>083</td>
<td>05/03</td>
<td>LATIN CAPITAL LETTER S</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>084</td>
<td>05/04</td>
<td>LATIN CAPITAL LETTER T</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>085</td>
<td>05/05</td>
<td>LATIN CAPITAL LETTER U</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>086</td>
<td>05/06</td>
<td>LATIN CAPITAL LETTER V</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>087</td>
<td>05/07</td>
<td>LATIN CAPITAL LETTER W</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>088</td>
<td>05/08</td>
<td>LATIN CAPITAL LETTER X</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>089</td>
<td>05/09</td>
<td>LATIN CAPITAL LETTER Y</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>090</td>
<td>05/10</td>
<td>LATIN CAPITAL LETTER Z</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>091</td>
<td>05/11</td>
<td>LEFT SQUARE BRACKET</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>092</td>
<td>05/12</td>
<td>REVERSE SOLIDUS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>093</td>
<td>05/13</td>
<td>RIGHT SQUARE BRACKET</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>094</td>
<td>05/14</td>
<td>CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>095</td>
<td>05/15</td>
<td>LOW LINE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>096</td>
<td>06/00</td>
<td>GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>097</td>
<td>06/01</td>
<td>LATIN SMALL LETTER a</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>098</td>
<td>06/02</td>
<td>LATIN SMALL LETTER b</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>099</td>
<td>06/03</td>
<td>LATIN SMALL LETTER c</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>100</td>
<td>06/04</td>
<td>LATIN SMALL LETTER d</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>101</td>
<td>06/05</td>
<td>LATIN SMALL LETTER e</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>102</td>
<td>06/06</td>
<td>LATIN SMALL LETTER f</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>103</td>
<td>06/07</td>
<td>LATIN SMALL LETTER g</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>104</td>
<td>06/08</td>
<td>LATIN SMALL LETTER h</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>105</td>
<td>06/09</td>
<td>LATIN SMALL LETTER i</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>106</td>
<td>06/10</td>
<td>LATIN SMALL LETTER j</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>107</td>
<td>06/11</td>
<td>LATIN SMALL LETTER k</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>108</td>
<td>06/12</td>
<td>LATIN SMALL LETTER l</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>109</td>
<td>06/13</td>
<td>LATIN SMALL LETTER m</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>110</td>
<td>06/14</td>
<td>LATIN SMALL LETTER n</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>111</td>
<td>06/15</td>
<td>LATIN SMALL LETTER o</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>112</td>
<td>07/00</td>
<td>LATIN SMALL LETTER p</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>113</td>
<td>07/01</td>
<td>LATIN SMALL LETTER q</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>114</td>
<td>07/02</td>
<td>LATIN SMALL LETTER r</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>115</td>
<td>07/03</td>
<td>LATIN SMALL LETTER s</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>116</td>
<td>07/04</td>
<td>LATIN SMALL LETTER t</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>117</td>
<td>07/05</td>
<td>LATIN SMALL LETTER u</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>118</td>
<td>07/06</td>
<td>LATIN SMALL LETTER v</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>119</td>
<td>07/07</td>
<td>LATIN SMALL LETTER w</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>120</td>
<td>07/08</td>
<td>LATIN SMALL LETTER x</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>121</td>
<td>07/09</td>
<td>LATIN SMALL LETTER y</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>122</td>
<td>07/10</td>
<td>LATIN SMALL LETTER z</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>123</td>
<td>07/11</td>
<td>LEFT CURLY BRACKET</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>124</td>
<td>07/12</td>
<td>VERTICAL LINE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>125</td>
<td>07/13</td>
<td>RIGHT CURLY BRACKET</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>126</td>
<td>07/14</td>
<td>TILDE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>127</td>
<td>07/15</td>
<td>NO-BREAK SPACE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>128</td>
<td>07/16</td>
<td>INVERTED EXCLAMATION MARK</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>129</td>
<td>07/17</td>
<td>CENT SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>130</td>
<td>07/18</td>
<td>POUND SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>131</td>
<td>07/19</td>
<td>CURRENCY SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>132</td>
<td>07/20</td>
<td>YEN SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>133</td>
<td>07/21</td>
<td>BROKEN VERTICAL BAR</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>134</td>
<td>07/22</td>
<td>SECTION SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>135</td>
<td>07/23</td>
<td>DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>136</td>
<td>07/24</td>
<td>COPYRIGHT SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>137</td>
<td>07/25</td>
<td>FEMININE ORDINAL INDICATOR</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>138</td>
<td>07/26</td>
<td>LEFT ANGLE QUOTATION MARK</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>139</td>
<td>07/27</td>
<td>NOT SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>140</td>
<td>07/28</td>
<td>HYPHEN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>141</td>
<td>07/29</td>
<td>REGISTERED TRADEMARK SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>142</td>
<td>07/30</td>
<td>MACRON</td>
<td>Latin-1</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Byte 4</td>
<td>Code</td>
<td>Name</td>
<td>Set</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>------</td>
<td>--------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>000</td>
<td>176</td>
<td>11/00</td>
<td>DEGREE SIGN, RING ABOVE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>177</td>
<td>11/01</td>
<td>PLUS-MINUS SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>178</td>
<td>11/02</td>
<td>SUPERSCRIPT TWO</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>179</td>
<td>11/03</td>
<td>SUPERSCRIPT THREE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>180</td>
<td>11/04</td>
<td>ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>181</td>
<td>11/05</td>
<td>MICRO SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>182</td>
<td>11/06</td>
<td>PARAGRAPH SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>183</td>
<td>11/07</td>
<td>MIDDLE DOT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>184</td>
<td>11/08</td>
<td>CEDILLA</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>185</td>
<td>11/09</td>
<td>SUPERSCRIPT ONE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>186</td>
<td>11/10</td>
<td>MASCULINE ORDINAL INDICATOR</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>187</td>
<td>11/11</td>
<td>RIGHT ANGLE QUOTATION MARK</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>188</td>
<td>11/12</td>
<td>VULGAR FRACTION ONE QUARTER</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>189</td>
<td>11/13</td>
<td>VULGAR FRACTION ONE HALF</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>190</td>
<td>11/14</td>
<td>VULGAR FRACTION THREE QUARTERS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>191</td>
<td>11/15</td>
<td>INVERTED QUESTION MARK</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>192</td>
<td>12/00</td>
<td>LATIN CAPITAL LETTER A WITH GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>193</td>
<td>12/01</td>
<td>LATIN CAPITAL LETTER A WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>194</td>
<td>12/02</td>
<td>LATIN CAPITAL LETTER A WITH CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>195</td>
<td>12/03</td>
<td>LATIN CAPITAL LETTER A WITH TILDE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>196</td>
<td>12/04</td>
<td>LATIN CAPITAL LETTER A WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>197</td>
<td>12/05</td>
<td>LATIN CAPITAL LETTER A WITH RING ABOVE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>198</td>
<td>12/06</td>
<td>LATIN CAPITAL DIPHTHONG AE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>199</td>
<td>12/07</td>
<td>LATIN CAPITAL LETTER C WITH CEDILLA</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>200</td>
<td>12/08</td>
<td>LATIN CAPITAL LETTER E WITH GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>201</td>
<td>12/09</td>
<td>LATIN CAPITAL LETTER E WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>202</td>
<td>12/10</td>
<td>LATIN CAPITAL LETTER E WITH CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>203</td>
<td>12/11</td>
<td>LATIN CAPITAL LETTER E WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>204</td>
<td>12/12</td>
<td>LATIN CAPITAL LETTER I WITH GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>205</td>
<td>12/13</td>
<td>LATIN CAPITAL LETTER I WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>206</td>
<td>12/14</td>
<td>LATIN CAPITAL LETTER I WITH CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>207</td>
<td>12/15</td>
<td>LATIN CAPITAL LETTER I WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>208</td>
<td>13/00</td>
<td>ICELANDIC CAPITAL LETTER ETH</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>209</td>
<td>13/01</td>
<td>LATIN CAPITAL LETTER N WITH TILDE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>210</td>
<td>13/02</td>
<td>LATIN CAPITAL LETTER O WITH GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>211</td>
<td>13/03</td>
<td>LATIN CAPITAL LETTER O WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>212</td>
<td>13/04</td>
<td>LATIN CAPITAL LETTER O WITH CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>213</td>
<td>13/05</td>
<td>LATIN CAPITAL LETTER O WITH TILDE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>214</td>
<td>13/06</td>
<td>LATIN CAPITAL LETTER O WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>215</td>
<td>13/07</td>
<td>MULTIPLICATION SIGN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>216</td>
<td>13/08</td>
<td>LATIN CAPITAL LETTER O WITH OBLIQUE STROKE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>217</td>
<td>13/09</td>
<td>LATIN CAPITAL LETTER U WITH GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>218</td>
<td>13/10</td>
<td>LATIN CAPITAL LETTER U WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>219</td>
<td>13/11</td>
<td>LATIN CAPITAL LETTER U WITH CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>220</td>
<td>13/12</td>
<td>LATIN CAPITAL LETTER U WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>221</td>
<td>13/13</td>
<td>LATIN CAPITAL LETTER Y WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>222</td>
<td>13/14</td>
<td>ICELANDIC CAPITAL LETTER THORN</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>223</td>
<td>13/15</td>
<td>GERMAN SMALL LETTER SHARP s</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>224</td>
<td>14/00</td>
<td>LATIN SMALL LETTER a WITH GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>225</td>
<td>14/01</td>
<td>LATIN SMALL LETTER a WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>226</td>
<td>14/02</td>
<td>LATIN SMALL LETTER a WITH CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>227</td>
<td>14/03</td>
<td>LATIN SMALL LETTER a WITH TILDE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>228</td>
<td>14/04</td>
<td>LATIN SMALL LETTER a WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>229</td>
<td>14/05</td>
<td>LATIN SMALL LETTER a WITH RING ABOVE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>230</td>
<td>14/06</td>
<td>LATIN SMALL DIPHTHONG ae</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>231</td>
<td>14/07</td>
<td>LATIN SMALL LETTER c WITH CEDILLA</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>232</td>
<td>14/08</td>
<td>LATIN SMALL LETTER e WITH GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>233</td>
<td>14/09</td>
<td>LATIN SMALL LETTER e WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>234</td>
<td>14/10</td>
<td>LATIN SMALL LETTER e WITH CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>235</td>
<td>14/11</td>
<td>LATIN SMALL LETTER e WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>000</td>
<td>236</td>
<td>14/12</td>
<td>LATIN SMALL LETTER i WITH GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
</tbody>
</table>

79
<table>
<thead>
<tr>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Code Pos</th>
<th>Name</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>237</td>
<td>238</td>
<td>14/13</td>
<td>LATIN SMALL LETTER i WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>239</td>
<td>240</td>
<td>14/15</td>
<td>LATIN SMALL LETTER i WITH CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>241</td>
<td>242</td>
<td>14/17</td>
<td>LATIN SMALL LETTER i WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>244</td>
<td>245</td>
<td>15/00</td>
<td>ICELANDIC SMALL LETTER ETH</td>
<td>Latin-1</td>
</tr>
<tr>
<td>248</td>
<td>249</td>
<td>15/01</td>
<td>LATIN SMALL LETTER n WITH TILDE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>252</td>
<td>253</td>
<td>15/03</td>
<td>LATIN SMALL LETTER o WITH GRAVE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>256</td>
<td>257</td>
<td>15/05</td>
<td>LATIN SMALL LETTER o WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>260</td>
<td>261</td>
<td>15/07</td>
<td>LATIN SMALL LETTER o WITH CIRCUMFLEX ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>264</td>
<td>265</td>
<td>15/09</td>
<td>LATIN SMALL LETTER o WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>268</td>
<td>270</td>
<td>15/11</td>
<td>LATIN SMALL LETTER o WITH TILDE</td>
<td>Latin-1</td>
</tr>
<tr>
<td>272</td>
<td>273</td>
<td>15/13</td>
<td>LATIN SMALL LETTER o WITH ACUTE ACCENT</td>
<td>Latin-1</td>
</tr>
<tr>
<td>276</td>
<td>277</td>
<td>15/15</td>
<td>LATIN SMALL LETTER o WITH DIAERESIS</td>
<td>Latin-1</td>
</tr>
<tr>
<td>281</td>
<td>282</td>
<td>16/01</td>
<td>LATIN SMALL LETTER a WITH O GONEK</td>
<td>Latin-2</td>
</tr>
<tr>
<td>285</td>
<td>286</td>
<td>16/02</td>
<td>BREVE</td>
<td>Latin-2</td>
</tr>
<tr>
<td>289</td>
<td>290</td>
<td>16/03</td>
<td>LATIN SMALL LETTER L WITH STROKE</td>
<td>Latin-2</td>
</tr>
<tr>
<td>293</td>
<td>294</td>
<td>16/04</td>
<td>LATIN CAPITAL LETTER A WITH OGONEK</td>
<td>Latin-2</td>
</tr>
<tr>
<td>298</td>
<td>299</td>
<td>16/05</td>
<td>LATIN CAPITAL LETTER L WITH CARON</td>
<td>Latin-2</td>
</tr>
<tr>
<td>303</td>
<td>304</td>
<td>16/06</td>
<td>LATIN CAPITAL LETTER S WITH ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>308</td>
<td>309</td>
<td>16/07</td>
<td>LATIN CAPITAL LETTER S WITH CARON</td>
<td>Latin-2</td>
</tr>
<tr>
<td>313</td>
<td>314</td>
<td>16/08</td>
<td>LATIN CAPITAL LETTER T WITH CARON</td>
<td>Latin-2</td>
</tr>
<tr>
<td>318</td>
<td>319</td>
<td>16/09</td>
<td>LATIN CAPITAL LETTER U WITH ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>323</td>
<td>324</td>
<td>16/10</td>
<td>LATIN CAPITAL LETTER U WITH DIAERESIS</td>
<td>Latin-2</td>
</tr>
<tr>
<td>328</td>
<td>329</td>
<td>16/11</td>
<td>LATIN CAPITAL LETTER U WITH TILDE</td>
<td>Latin-2</td>
</tr>
<tr>
<td>334</td>
<td>335</td>
<td>16/12</td>
<td>LATIN CAPITAL LETTER U WITH ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>339</td>
<td>340</td>
<td>16/13</td>
<td>LATIN CAPITAL LETTER U WITH DIAERESIS</td>
<td>Latin-2</td>
</tr>
<tr>
<td>345</td>
<td>346</td>
<td>16/14</td>
<td>LATIN CAPITAL LETTER U WITH TILDE</td>
<td>Latin-2</td>
</tr>
<tr>
<td>350</td>
<td>351</td>
<td>16/15</td>
<td>LATIN CAPITAL LETTER U WITH ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>355</td>
<td>356</td>
<td>16/16</td>
<td>LATIN CAPITAL LETTER U WITH DIAERESIS</td>
<td>Latin-2</td>
</tr>
</tbody>
</table>

---

000 001 161 10/01 LATIN CAPITAL LETTER A WITH OGONEK Latin-2
000 001 162 10/02 BREVE Latin-2
000 001 163 10/03 LATIN CAPITAL LETTER L WITH STROKE Latin-2
000 001 165 10/05 LATIN CAPITAL LETTER L WITH CARON Latin-2
000 001 166 10/06 LATIN CAPITAL LETTER S WITH ACUTE ACCENT Latin-2
000 001 169 10/09 LATIN CAPITAL LETTER S WITH CARON Latin-2
000 001 170 10/10 LATIN CAPITAL LETTER S WITH CEDILLA Latin-2
000 001 171 10/11 LATIN CAPITAL LETTER T WITH CARON Latin-2
000 001 172 10/12 LATIN CAPITAL LETTER Z WITH ACUTE ACCENT Latin-2
000 001 174 10/14 LATIN CAPITAL LETTER Z WITH CARON Latin-2
000 001 175 10/15 LATIN CAPITAL LETTER Z WITH DOT ABOVE Latin-2
000 001 177 11/01 LATIN SMALL LETTER a WITH O GONEK Latin-2
000 001 178 11/02 OGONEK Latin-2
000 001 179 11/03 LATIN SMALL LETTER i WITH STROKE Latin-2
000 001 181 11/05 LATIN SMALL LETTER i WITH CARON Latin-2
000 001 182 11/06 LATIN SMALL LETTER s WITH ACUTE ACCENT Latin-2
000 001 183 11/07 CARON Latin-2
000 001 185 11/09 LATIN SMALL LETTER s WITH CARON Latin-2
000 001 186 11/10 LATIN SMALL LETTER s WITH CEDILLA Latin-2
000 001 187 11/11 LATIN SMALL LETTER t WITH CARON Latin-2
000 001 188 11/12 LATIN SMALL LETTER z WITH ACUTE ACCENT Latin-2
000 001 189 11/13 DOUBLE ACUTE ACCENT Latin-2
000 001 190 11/14 LATIN SMALL LETTER z WITH CARON Latin-2
000 001 191 11/15 LATIN SMALL LETTER z WITH DOT ABOVE Latin-2
000 001 192 12/00 LATIN CAPITAL LETTER R WITH ACUTE ACCENT Latin-2
000 001 195 12/03 LATIN CAPITAL LETTER A WITH BREVE Latin-2
000 001 197 12/05 LATIN CAPITAL LETTER L WITH ACUTE ACCENT Latin-2
000 001 198 12/06 LATIN CAPITAL LETTER C WITH ACUTE ACCENT Latin-2
000 001 200 12/08 LATIN CAPITAL LETTER C WITH CARON Latin-2
000 001 202 12/10 LATIN CAPITAL LETTER E WITH OGONEK Latin-2
000 001 204 12/12 LATIN CAPITAL LETTER E WITH CARON Latin-2
000 001 207 12/15 LATIN CAPITAL LETTER D WITH CARON Latin-2
000 001 208 13/00 LATIN CAPITAL LETTER D WITH STROKE Latin-2
000 001 209 13/01 LATIN CAPITAL LETTER N WITH ACUTE ACCENT Latin-2
000 001 210 13/02 LATIN CAPITAL LETTER N WITH CARON Latin-2
000 001 213 13/05 LATIN CAPITAL LETTER O WITH DOUBLE ACUTE ACCENT Latin-2
000 001 216 13/08 LATIN CAPITAL LETTER R WITH CARON Latin-2
000 001 217 13/09 LATIN CAPITAL LETTER U WITH RING ABOVE Latin-2
000 001 219 13/11 LATIN CAPITAL LETTER U WITH DOUBLE ACUTE ACCENT Latin-2
000 001 222 13/14 LATIN CAPITAL LETTER T WITH CEDILLA Latin-2
<table>
<thead>
<tr>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Code Pos</th>
<th>Name</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>224</td>
<td>14/00</td>
<td>LATIN SMALL LETTER r WITH ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>227</td>
<td>14/03</td>
<td>LATIN SMALL LETTER a WITH BREVE</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>229</td>
<td>14/05</td>
<td>LATIN SMALL LETTER i WITH ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>230</td>
<td>14/06</td>
<td>LATIN SMALL LETTER c WITH ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>232</td>
<td>14/08</td>
<td>LATIN SMALL LETTER e WITH CARON</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>234</td>
<td>14/10</td>
<td>LATIN SMALL LETTER e WITH OGONEK</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>236</td>
<td>14/12</td>
<td>LATIN SMALL LETTER e WITH CARON</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>239</td>
<td>14/15</td>
<td>LATIN SMALL LETTER d WITH CARON</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>240</td>
<td>15/00</td>
<td>LATIN SMALL LETTER d WITH STROKE</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>241</td>
<td>15/01</td>
<td>LATIN SMALL LETTER n WITH ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>242</td>
<td>15/02</td>
<td>LATIN SMALL LETTER n WITH CARON</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>245</td>
<td>15/05</td>
<td>LATIN SMALL LETTER o WITH DOUBLE ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>248</td>
<td>15/08</td>
<td>LATIN SMALL LETTER r WITH CARON</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>249</td>
<td>15/09</td>
<td>LATIN SMALL LETTER u WITH RING ABOVE</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>251</td>
<td>15/11</td>
<td>LATIN SMALL LETTER u WITH DOUBLE ACUTE ACCENT</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>254</td>
<td>15/14</td>
<td>LATIN SMALL LETTER t WITH CEDILLA</td>
<td>Latin-2</td>
</tr>
<tr>
<td>001</td>
<td>255</td>
<td>15/15</td>
<td>DOT ABOVE</td>
<td>Latin-2</td>
</tr>
<tr>
<td>002</td>
<td>161</td>
<td>10/01</td>
<td>LATIN CAPITAL LETTER H WITH STROKE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>166</td>
<td>10/06</td>
<td>LATIN CAPITAL LETTER H WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>169</td>
<td>10/09</td>
<td>LATIN CAPITAL LETTER I WITH DOT ABOVE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>171</td>
<td>10/11</td>
<td>LATIN CAPITAL LETTER G WITH BREVE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>172</td>
<td>10/12</td>
<td>LATIN CAPITAL LETTER J WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>177</td>
<td>11/01</td>
<td>LATIN SMALL LETTER h WITH STROKE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>182</td>
<td>11/06</td>
<td>LATIN SMALL LETTER h WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>183</td>
<td>11/09</td>
<td>SMALL DOTLESS LETTER i</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>187</td>
<td>11/11</td>
<td>LATIN SMALL LETTER g WITH BREVE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>188</td>
<td>11/12</td>
<td>LATIN SMALL LETTER j WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>197</td>
<td>12/05</td>
<td>LATIN CAPITAL LETTER C WITH DOT ABOVE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>198</td>
<td>12/06</td>
<td>LATIN CAPITAL LETTER C WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>213</td>
<td>13/05</td>
<td>LATIN CAPITAL LETTER G WITH DOT ABOVE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>216</td>
<td>13/08</td>
<td>LATIN CAPITAL LETTER G WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>221</td>
<td>13/13</td>
<td>LATIN CAPITAL LETTER U WITH BREVE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>222</td>
<td>13/14</td>
<td>LATIN CAPITAL LETTER S WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>229</td>
<td>14/05</td>
<td>LATIN SMALL LETTER c WITH DOT ABOVE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>230</td>
<td>14/06</td>
<td>LATIN SMALL LETTER c WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>245</td>
<td>15/05</td>
<td>LATIN SMALL LETTER g WITH DOT ABOVE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>248</td>
<td>15/08</td>
<td>LATIN SMALL LETTER g WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>253</td>
<td>15/13</td>
<td>LATIN SMALL LETTER u WITH BREVE</td>
<td>Latin-3</td>
</tr>
<tr>
<td>002</td>
<td>254</td>
<td>15/14</td>
<td>LATIN SMALL LETTER s WITH CIRCUMFLEX ACCENT</td>
<td>Latin-3</td>
</tr>
<tr>
<td>003</td>
<td>162</td>
<td>10/02</td>
<td>SMALL GREENLANDIC LETTER KRA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>163</td>
<td>10/03</td>
<td>LATIN CAPITAL LETTER R WITH CEDILLA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>165</td>
<td>10/05</td>
<td>LATIN CAPITAL LETTER I WITH TILDE</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>166</td>
<td>10/06</td>
<td>LATIN CAPITAL LETTER L WITH CEDILLA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>170</td>
<td>10/10</td>
<td>LATIN CAPITAL LETTER E WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>171</td>
<td>10/11</td>
<td>LATIN CAPITAL LETTER G WITH CEDILLA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>172</td>
<td>10/12</td>
<td>LATIN CAPITAL LETTER T WITH OBlique STROKE</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>179</td>
<td>11/03</td>
<td>LATIN SMALL LETTER r WITH CEDILLA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>181</td>
<td>11/05</td>
<td>LATIN SMALL LETTER i WITH TILDE</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>182</td>
<td>11/06</td>
<td>LATIN SMALL LETTER i WITH CEDILLA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>186</td>
<td>11/10</td>
<td>LATIN SMALL LETTER e WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>187</td>
<td>11/11</td>
<td>LATIN SMALL LETTER g WITH CEDILLA ABOVE</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>188</td>
<td>11/12</td>
<td>LATIN SMALL LETTER t WITH OBlique STROKE</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>189</td>
<td>11/13</td>
<td>LAPPISH CAPITAL LETTER ENG</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>191</td>
<td>11/15</td>
<td>LAPPISH SMALL LETTER ENG</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>192</td>
<td>12/00</td>
<td>LATIN CAPITAL LETTER A WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>199</td>
<td>12/07</td>
<td>LATIN CAPITAL LETTER I WITH OGONEK</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>204</td>
<td>12/12</td>
<td>LATIN CAPITAL LETTER E WITH DOT ABOVE</td>
<td>Latin-4</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Byte 4</td>
<td>Code Pos</td>
<td>Name</td>
<td>Set</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>----------</td>
<td>-------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>003</td>
<td>207</td>
<td>12/15</td>
<td>LATIN CAPITAL LETTER I WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>209</td>
<td>13/01</td>
<td>LATIN CAPITAL LETTER N WITH CEDILLA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>210</td>
<td>13/02</td>
<td>LATIN CAPITAL LETTER O WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>211</td>
<td>13/03</td>
<td>LATIN CAPITAL LETTER K WITH CEDILLA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>217</td>
<td>13/09</td>
<td>LATIN CAPITAL LETTER U WITH OGONEK</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>221</td>
<td>13/13</td>
<td>LATIN CAPITAL LETTER U WITH TILDE</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>222</td>
<td>13/14</td>
<td>LATIN CAPITAL LETTER U WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>224</td>
<td>14/00</td>
<td>LATIN SMALL LETTER a WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>231</td>
<td>14/07</td>
<td>LATIN SMALL LETTER i WITH OGONEK</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>236</td>
<td>14/12</td>
<td>LATIN SMALL LETTER e WITH DOT ABOVE</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>239</td>
<td>14/15</td>
<td>LATIN SMALL LETTER i WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>241</td>
<td>15/01</td>
<td>LATIN SMALL LETTER n WITH CEDILLA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>242</td>
<td>15/02</td>
<td>LATIN SMALL LETTER o WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>243</td>
<td>15/03</td>
<td>LATIN SMALL LETTER k WITH CEDILLA</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>249</td>
<td>15/09</td>
<td>LATIN SMALL LETTER u WITH OGONEK</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>253</td>
<td>15/13</td>
<td>LATIN SMALL LETTER u WITH TILDE</td>
<td>Latin-4</td>
</tr>
<tr>
<td>003</td>
<td>254</td>
<td>15/14</td>
<td>LATIN SMALL LETTER u WITH MACRON</td>
<td>Latin-4</td>
</tr>
<tr>
<td>004</td>
<td>126</td>
<td>07/14</td>
<td>OVERLINE</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>161</td>
<td>10/01</td>
<td>KANA FULL STOP</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>162</td>
<td>10/02</td>
<td>KANA OPENING BRACKET</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>163</td>
<td>10/03</td>
<td>KANA CLOSING BRACKET</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>164</td>
<td>10/04</td>
<td>KANA COMMA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>165</td>
<td>10/05</td>
<td>KANA Conjunctive</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>166</td>
<td>10/06</td>
<td>KANA LETTER WO</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>167</td>
<td>10/07</td>
<td>KANA LETTER SMALL A</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>168</td>
<td>10/08</td>
<td>KANA LETTER SMALL I</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>169</td>
<td>10/09</td>
<td>KANA LETTER SMALL U</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>170</td>
<td>10/10</td>
<td>KANA LETTER SMALL E</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>171</td>
<td>10/11</td>
<td>KANA LETTER SMALL O</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>172</td>
<td>10/12</td>
<td>KANA LETTER SMALL YA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>173</td>
<td>10/13</td>
<td>KANA LETTER SMALL YU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>174</td>
<td>10/14</td>
<td>KANA LETTER SMALL YO</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>175</td>
<td>10/15</td>
<td>KANA LETTER SMALL TSU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>176</td>
<td>11/00</td>
<td>PROLONGED SOUND SYMBOL</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>177</td>
<td>11/01</td>
<td>KANA LETTER A</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>178</td>
<td>11/02</td>
<td>KANA LETTER I</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>179</td>
<td>11/03</td>
<td>KANA LETTER U</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>180</td>
<td>11/04</td>
<td>KANA LETTER E</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>181</td>
<td>11/05</td>
<td>KANA LETTER O</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>182</td>
<td>11/06</td>
<td>KANA LETTER KA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>183</td>
<td>11/07</td>
<td>KANA LETTER KI</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>184</td>
<td>11/08</td>
<td>KANA LETTER KU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>185</td>
<td>11/09</td>
<td>KANA LETTER KE</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>186</td>
<td>11/10</td>
<td>KANA LETTER KO</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>187</td>
<td>11/11</td>
<td>KANA LETTER SA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>188</td>
<td>11/12</td>
<td>KANA LETTER SHI</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>189</td>
<td>11/13</td>
<td>KANA LETTER SU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>190</td>
<td>11/14</td>
<td>KANA LETTER SE</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>191</td>
<td>11/15</td>
<td>KANA LETTER SO</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>192</td>
<td>12/00</td>
<td>KANA LETTER TA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>193</td>
<td>12/01</td>
<td>KANA LETTER CHI</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>194</td>
<td>12/02</td>
<td>KANA LETTER TSU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>195</td>
<td>12/03</td>
<td>KANA LETTER TE</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>196</td>
<td>12/04</td>
<td>KANA LETTER TO</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>197</td>
<td>12/05</td>
<td>KANA LETTER NA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>198</td>
<td>12/06</td>
<td>KANA LETTER NI</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>199</td>
<td>12/07</td>
<td>KANA LETTER NU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>200</td>
<td>12/08</td>
<td>KANA LETTER NE</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>201</td>
<td>12/09</td>
<td>KANA LETTER NO</td>
<td>Kana</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Byte 4</td>
<td>Code Pos</td>
<td>Name</td>
<td>Set</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>----------</td>
<td>----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>004</td>
<td>202</td>
<td>12/10</td>
<td>KANA LETTER HA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>203</td>
<td>12/11</td>
<td>KANA LETTER HI</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>204</td>
<td>12/12</td>
<td>KANA LETTER FU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>205</td>
<td>12/13</td>
<td>KANA LETTER HE</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>206</td>
<td>12/14</td>
<td>KANA LETTER HO</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>207</td>
<td>12/15</td>
<td>KANA LETTER MA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>208</td>
<td>13/00</td>
<td>KANA LETTER MI</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>209</td>
<td>13/01</td>
<td>KANA LETTER MU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>210</td>
<td>13/02</td>
<td>KANA LETTER ME</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>211</td>
<td>13/03</td>
<td>KANA LETTER MO</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>212</td>
<td>13/04</td>
<td>KANA LETTER YA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>213</td>
<td>13/05</td>
<td>KANA LETTER YU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>214</td>
<td>13/06</td>
<td>KANA LETTER YO</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>215</td>
<td>13/07</td>
<td>KANA LETTER RA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>216</td>
<td>13/08</td>
<td>KANA LETTER RI</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>217</td>
<td>13/09</td>
<td>KANA LETTER RU</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>218</td>
<td>13/10</td>
<td>KANA LETTER RE</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>219</td>
<td>13/11</td>
<td>KANA LETTER RO</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>220</td>
<td>13/12</td>
<td>KANA LETTER WA</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>221</td>
<td>13/13</td>
<td>KANA LETTER N</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>222</td>
<td>13/14</td>
<td>VOICED SOUND SYMBOL</td>
<td>Kana</td>
</tr>
<tr>
<td>004</td>
<td>223</td>
<td>13/15</td>
<td>SEMIVOICED SOUND SYMBOL</td>
<td>Kana</td>
</tr>
<tr>
<td>005</td>
<td>172</td>
<td>10/12</td>
<td>ARABIC COMMA</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>187</td>
<td>11/11</td>
<td>ARABIC SEMICOLON</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>191</td>
<td>11/15</td>
<td>ARABIC QUESTION MARK</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>193</td>
<td>12/01</td>
<td>ARABIC LETTER HAMZA</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>194</td>
<td>12/02</td>
<td>ARABIC LETTER MADDA ON ALEF</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>195</td>
<td>12/03</td>
<td>ARABIC LETTER HAMZA ON ALEF</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>196</td>
<td>12/04</td>
<td>ARABIC LETTER HAMZA ON WAW</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>197</td>
<td>12/05</td>
<td>ARABIC LETTER HAMZA UNDER ALEF</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>198</td>
<td>12/06</td>
<td>ARABIC LETTER HAMZA ON YEH</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>199</td>
<td>12/07</td>
<td>ARABIC LETTER ALEF</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>200</td>
<td>12/08</td>
<td>ARABIC LETTER BEH</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>201</td>
<td>12/09</td>
<td>ARABIC LETTER TEH MARBUTA</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>202</td>
<td>12/10</td>
<td>ARABIC LETTER TEH</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>203</td>
<td>12/11</td>
<td>ARABIC LETTER THEH</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>204</td>
<td>12/12</td>
<td>ARABIC LETTER JEEM</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>205</td>
<td>12/13</td>
<td>ARABIC LETTER HAH</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>206</td>
<td>12/14</td>
<td>ARABIC LETTER KHAI</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>207</td>
<td>12/15</td>
<td>ARABIC LETTER DAL</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>208</td>
<td>13/00</td>
<td>ARABIC LETTER THAL</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>209</td>
<td>13/01</td>
<td>ARABIC LETTER RA</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>210</td>
<td>13/02</td>
<td>ARABIC LETTER ZAIN</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>211</td>
<td>13/03</td>
<td>ARABIC LETTER SEEN</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>212</td>
<td>13/04</td>
<td>ARABIC LETTER SHEEN</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>213</td>
<td>13/05</td>
<td>ARABIC LETTER SAD</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>214</td>
<td>13/06</td>
<td>ARABIC LETTER DAD</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>215</td>
<td>13/07</td>
<td>ARABIC LETTER TAH</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>216</td>
<td>13/08</td>
<td>ARABIC LETTER ZAH</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>217</td>
<td>13/09</td>
<td>ARABIC LETTER AIN</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>218</td>
<td>13/10</td>
<td>ARABIC LETTER GHAIN</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>224</td>
<td>14/00</td>
<td>ARABIC LETTER TATWEEL</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>225</td>
<td>14/01</td>
<td>ARABIC LETTER FEH</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>226</td>
<td>14/02</td>
<td>ARABIC LETTER QAF</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>227</td>
<td>14/03</td>
<td>ARABIC LETTER KAF</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>228</td>
<td>14/04</td>
<td>ARABIC LETTER LAM</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>229</td>
<td>14/05</td>
<td>ARABIC LETTER MEEM</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>230</td>
<td>14/06</td>
<td>ARABIC LETTER NOON</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>231</td>
<td>14/07</td>
<td>ARABIC LETTER HA</td>
<td>Arabic</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Byte 4</td>
<td>Code Pos</td>
<td>Name</td>
<td>Set</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>----------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>005</td>
<td>232</td>
<td>14/08</td>
<td>ARABIC LETTER WAW</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>233</td>
<td>14/09</td>
<td>ARABIC LETTER ALEF MAKURA</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>234</td>
<td>14/10</td>
<td>ARABIC LETTER YEH</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>235</td>
<td>14/11</td>
<td>ARABIC LETTER FATHATAN</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>236</td>
<td>14/12</td>
<td>ARABIC LETTER DAMMATAN</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>237</td>
<td>14/13</td>
<td>ARABIC LETTER KASRATAN</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>238</td>
<td>14/14</td>
<td>ARABIC LETTER FATHA</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>239</td>
<td>14/15</td>
<td>ARABIC LETTER DAMMA</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>240</td>
<td>15/00</td>
<td>ARABIC LETTER KASRA</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>241</td>
<td>15/01</td>
<td>ARABIC LETTER SHADDA</td>
<td>Arabic</td>
</tr>
<tr>
<td>005</td>
<td>242</td>
<td>15/02</td>
<td>ARABIC LETTER SUKUN</td>
<td>Arabic</td>
</tr>
<tr>
<td>006</td>
<td>161</td>
<td>10/01</td>
<td>SERBOCROATION CYRILLIC SMALL LETTER DJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>162</td>
<td>10/02</td>
<td>MACEDONIAN CYRILLIC SMALL LETTER GJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>163</td>
<td>10/03</td>
<td>CYRILLIC SMALL LETTER IO</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>164</td>
<td>10/04</td>
<td>UKRAINIAN CYRILLIC SMALL LETTER IE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>165</td>
<td>10/05</td>
<td>MACEDONIAN SMALL LETTER DSE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>166</td>
<td>10/06</td>
<td>BYELORUSSIAN/UKRAINIAN CYRILLIC SMALL LETTER I</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>167</td>
<td>10/07</td>
<td>UKRAINIAN SMALL LETTER YI</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>168</td>
<td>10/08</td>
<td>CYRILLIC SMALL LETTER JE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>169</td>
<td>10/09</td>
<td>CYRILLIC SMALL LETTER LJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>170</td>
<td>10/10</td>
<td>CYRILLIC SMALL LETTER NJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>171</td>
<td>10/11</td>
<td>SERBIAN SMALL LETTER TSHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>172</td>
<td>10/12</td>
<td>MACEDONIAN CYRILLIC SMALL LETTER KJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>174</td>
<td>10/14</td>
<td>BYELORUSSIAN SMALL LETTER SHORT U</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>175</td>
<td>10/15</td>
<td>CYRILLIC SMALL LETTER DZHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>176</td>
<td>11/00</td>
<td>NUMERO SIGN</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>177</td>
<td>11/01</td>
<td>SERBOCROATION CYRILLIC CAPITAL LETTER DJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>178</td>
<td>11/02</td>
<td>MACEDONIAN CYRILLIC CAPITAL LETTER GJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>179</td>
<td>11/03</td>
<td>CYRILLIC CAPITAL LETTER IO</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>180</td>
<td>11/04</td>
<td>UKRAINIAN CYRILLIC CAPITAL LETTER IE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>181</td>
<td>11/05</td>
<td>MACEDONIAN CAPITAL LETTER DSE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>182</td>
<td>11/06</td>
<td>BYELORUSSIAN/UKRAINIAN CYRILLIC CAPITAL LETTER I</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>183</td>
<td>11/07</td>
<td>UKRAINIAN CAPITAL LETTER YI</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>184</td>
<td>11/08</td>
<td>CYRILLIC CAPITAL LETTER JE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>185</td>
<td>11/09</td>
<td>CYRILLIC CAPITAL LETTER LJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>186</td>
<td>11/10</td>
<td>CYRILLIC CAPITAL LETTER NJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>187</td>
<td>11/11</td>
<td>SERBIAN CAPITAL LETTER TSHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>188</td>
<td>11/12</td>
<td>MACEDONIAN CYRILLIC CAPITAL LETTER KJE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>190</td>
<td>11/14</td>
<td>BYELORUSSIAN CAPITAL LETTER SHORT U</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>191</td>
<td>11/15</td>
<td>CYRILLIC CAPITAL LETTER DZHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>192</td>
<td>12/00</td>
<td>CYRILLIC SMALL LETTER YU</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>193</td>
<td>12/01</td>
<td>CYRILLIC SMALL LETTER A</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>194</td>
<td>12/02</td>
<td>CYRILLIC SMALL LETTER E</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>195</td>
<td>12/03</td>
<td>CYRILLIC SMALL LETTER TSE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>196</td>
<td>12/04</td>
<td>CYRILLIC SMALL LETTER DE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>197</td>
<td>12/05</td>
<td>CYRILLIC SMALL LETTER IE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>198</td>
<td>12/06</td>
<td>CYRILLIC SMALL LETTER EF</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>199</td>
<td>12/07</td>
<td>CYRILLIC SMALL LETTER GHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>200</td>
<td>12/08</td>
<td>CYRILLIC SMALL LETTER HA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>201</td>
<td>12/09</td>
<td>CYRILLIC SMALL LETTER I</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>202</td>
<td>12/10</td>
<td>CYRILLIC SMALL LETTER SHORT I</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>203</td>
<td>12/11</td>
<td>CYRILLIC SMALL LETTER KA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>204</td>
<td>12/12</td>
<td>CYRILLIC SMALL LETTER EL</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>205</td>
<td>12/13</td>
<td>CYRILLIC SMALL LETTER EM</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>206</td>
<td>12/14</td>
<td>CYRILLIC SMALL LETTER EN</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>207</td>
<td>12/15</td>
<td>CYRILLIC SMALL LETTER O</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>208</td>
<td>13/00</td>
<td>CYRILLIC SMALL LETTER PE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>209</td>
<td>13/01</td>
<td>CYRILLIC SMALL LETTER YA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>210</td>
<td>13/02</td>
<td>CYRILLIC SMALL LETTER ER</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Byte 4</td>
<td>Code Pos</td>
<td>Name</td>
<td>Set</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>----------</td>
<td>-------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>006</td>
<td>211</td>
<td>13/03</td>
<td>CYRILLIC SMALL LETTER ES</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>212</td>
<td>13/04</td>
<td>CYRILLIC SMALL LETTER TE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>213</td>
<td>13/05</td>
<td>CYRILLIC SMALL LETTER U</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>214</td>
<td>13/06</td>
<td>CYRILLIC SMALL LETTER ZHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>215</td>
<td>13/07</td>
<td>CYRILLIC SMALL LETTER VE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>216</td>
<td>13/08</td>
<td>CYRILLIC SMALL SOFT SIGN</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>217</td>
<td>13/09</td>
<td>CYRILLIC SMALL LETTER YERU</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>218</td>
<td>13/10</td>
<td>CYRILLIC SMALL LETTER ZE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>219</td>
<td>13/11</td>
<td>CYRILLIC SMALL LETTER SHA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>220</td>
<td>13/12</td>
<td>CYRILLIC SMALL LETTER E</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>221</td>
<td>13/13</td>
<td>CYRILLIC SMALL LETTER SHCHA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>222</td>
<td>13/14</td>
<td>CYRILLIC SMALL LETTER CHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>223</td>
<td>13/15</td>
<td>CYRILLIC SMALL HARD SIGN</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>224</td>
<td>14/00</td>
<td>CYRILLIC CAPITAL LETTER YU</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>225</td>
<td>14/01</td>
<td>CYRILLIC CAPITAL LETTER A</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>226</td>
<td>14/02</td>
<td>CYRILLIC CAPITAL LETTER BE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>227</td>
<td>14/03</td>
<td>CYRILLIC CAPITAL LETTER TSE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>228</td>
<td>14/04</td>
<td>CYRILLIC CAPITAL LETTER DE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>229</td>
<td>14/05</td>
<td>CYRILLIC CAPITAL LETTER IE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>230</td>
<td>14/06</td>
<td>CYRILLIC CAPITAL LETTER EF</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>231</td>
<td>14/07</td>
<td>CYRILLIC CAPITAL LETTER GHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>232</td>
<td>14/08</td>
<td>CYRILLIC CAPITAL LETTER HA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>233</td>
<td>14/09</td>
<td>CYRILLIC CAPITAL LETTER I</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>234</td>
<td>14/10</td>
<td>CYRILLIC CAPITAL LETTER SHORT I</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>235</td>
<td>14/11</td>
<td>CYRILLIC CAPITAL LETTER KA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>236</td>
<td>14/12</td>
<td>CYRILLIC CAPITAL LETTER EL</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>237</td>
<td>14/13</td>
<td>CYRILLIC CAPITAL LETTER EM</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>238</td>
<td>14/14</td>
<td>CYRILLIC CAPITAL LETTER EN</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>239</td>
<td>14/15</td>
<td>CYRILLIC CAPITAL LETTER O</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>240</td>
<td>15/00</td>
<td>CYRILLIC CAPITAL LETTER PE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>241</td>
<td>15/01</td>
<td>CYRILLIC CAPITAL LETTER YA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>242</td>
<td>15/02</td>
<td>CYRILLIC CAPITAL LETTER ER</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>243</td>
<td>15/03</td>
<td>CYRILLIC CAPITAL LETTER ES</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>244</td>
<td>15/04</td>
<td>CYRILLIC CAPITAL LETTER TE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>245</td>
<td>15/05</td>
<td>CYRILLIC CAPITAL LETTER U</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>246</td>
<td>15/06</td>
<td>CYRILLIC CAPITAL LETTER ZHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>247</td>
<td>15/07</td>
<td>CYRILLIC CAPITAL LETTER VE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>248</td>
<td>15/08</td>
<td>CYRILLIC CAPITAL SOFT SIGN</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>249</td>
<td>15/09</td>
<td>CYRILLIC CAPITAL LETTER YERU</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>250</td>
<td>15/10</td>
<td>CYRILLIC CAPITAL LETTER ZE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>251</td>
<td>15/11</td>
<td>CYRILLIC CAPITAL LETTER SHA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>252</td>
<td>15/12</td>
<td>CYRILLIC CAPITAL LETTER E</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>253</td>
<td>15/13</td>
<td>CYRILLIC CAPITAL LETTER SHCHA</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>254</td>
<td>15/14</td>
<td>CYRILLIC CAPITAL LETTER CHE</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>006</td>
<td>255</td>
<td>15/15</td>
<td>CYRILLIC CAPITAL HARD SIGN</td>
<td>Cyrillic</td>
</tr>
<tr>
<td>007</td>
<td>161</td>
<td>10/01</td>
<td>GREEK CAPITAL LETTER ALPHA WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>162</td>
<td>10/02</td>
<td>GREEK CAPITAL LETTER EPSILON WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>163</td>
<td>10/03</td>
<td>GREEK CAPITAL LETTER ETA WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>164</td>
<td>10/04</td>
<td>GREEK CAPITAL LETTER IOTA WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>165</td>
<td>10/05</td>
<td>GREEK CAPITAL LETTER IOTA WITH DIAERESIS</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>167</td>
<td>10/07</td>
<td>GREEK CAPITAL LETTER OMICRON WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>168</td>
<td>10/08</td>
<td>GREEK CAPITAL LETTER UPSILON WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>169</td>
<td>10/09</td>
<td>GREEK CAPITAL LETTER UPSILON WITH DIAERESIS</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>171</td>
<td>10/11</td>
<td>GREEK CAPITAL LETTER OMEGA WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>174</td>
<td>10/14</td>
<td>DIAERESIS AND ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>175</td>
<td>10/15</td>
<td>HORIZONTAL BAR</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>177</td>
<td>11/01</td>
<td>GREEK SMALL LETTER ALPHA WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>178</td>
<td>11/02</td>
<td>GREEK SMALL LETTER EPSILON WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>179</td>
<td>11/03</td>
<td>GREEK SMALL LETTER ETA WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Byte 4</td>
<td>Code Pos</td>
<td>Name</td>
<td>Set</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>----------</td>
<td>-----------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>007</td>
<td>180</td>
<td>11/04</td>
<td>GREEK SMALL LETTER IOTA WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>181</td>
<td>11/05</td>
<td>GREEK SMALL LETTER IOTA WITH DIAERESIS</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>182</td>
<td>11/06</td>
<td>GREEK SMALL LETTER IOTA WITH ACCENT+DIAERESIS</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>183</td>
<td>11/07</td>
<td>GREEK SMALL LETTER OMICRON WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>184</td>
<td>11/08</td>
<td>GREEK SMALL LETTER UPSILON WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>185</td>
<td>11/09</td>
<td>GREEK SMALL LETTER UPSILON WITH DIAERESIS</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>186</td>
<td>11/10</td>
<td>GREEK SMALL LETTER UPSILON WITH ACCENT+DIAERESIS</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>187</td>
<td>11/11</td>
<td>GREEK SMALL LETTER OMEGA WITH ACCENT</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>193</td>
<td>12/01</td>
<td>GREEK CAPITAL LETTER ALPHA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>194</td>
<td>12/02</td>
<td>GREEK CAPITAL LETTER BETA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>195</td>
<td>12/03</td>
<td>GREEK CAPITAL LETTER GAMMA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>196</td>
<td>12/04</td>
<td>GREEK CAPITAL LETTER DELTA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>197</td>
<td>12/05</td>
<td>GREEK CAPITAL LETTER EPSILON</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>198</td>
<td>12/06</td>
<td>GREEK CAPITAL LETTER ZETA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>199</td>
<td>12/07</td>
<td>GREEK CAPITAL LETTER ETA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>200</td>
<td>12/08</td>
<td>GREEK CAPITAL LETTER THETA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>201</td>
<td>12/09</td>
<td>GREEK CAPITAL LETTER IOTA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>202</td>
<td>12/10</td>
<td>GREEK CAPITAL LETTER KAPPA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>203</td>
<td>12/11</td>
<td>GREEK CAPITAL LETTER LAMDA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>204</td>
<td>12/12</td>
<td>GREEK CAPITAL LETTER MU</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>205</td>
<td>12/13</td>
<td>GREEK CAPITAL LETTER NU</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>206</td>
<td>12/14</td>
<td>GREEK CAPITAL LETTER XI</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>207</td>
<td>12/15</td>
<td>GREEK CAPITAL LETTER OOMICRON</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>208</td>
<td>13/00</td>
<td>GREEK CAPITAL LETTER Pi</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>209</td>
<td>13/01</td>
<td>GREEK CAPITAL LETTER RHO</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>210</td>
<td>13/02</td>
<td>GREEK CAPITAL LETTER SIGMA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>212</td>
<td>13/04</td>
<td>GREEK CAPITAL LETTER TAU</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>213</td>
<td>13/05</td>
<td>GREEK CAPITAL LETTER UPSILON</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>214</td>
<td>13/06</td>
<td>GREEK CAPITAL LETTER PHI</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>215</td>
<td>13/07</td>
<td>GREEK CAPITAL LETTER CHI</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>216</td>
<td>13/08</td>
<td>GREEK CAPITAL LETTER PSI</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>217</td>
<td>13/09</td>
<td>GREEK CAPITAL LETTER OMEGA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>225</td>
<td>14/01</td>
<td>GREEK SMALL LETTER ALPHA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>226</td>
<td>14/02</td>
<td>GREEK SMALL LETTER BETA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>227</td>
<td>14/03</td>
<td>GREEK SMALL LETTER GAMMA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>228</td>
<td>14/04</td>
<td>GREEK SMALL LETTER DELTA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>229</td>
<td>14/05</td>
<td>GREEK SMALL LETTER EPSILON</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>230</td>
<td>14/06</td>
<td>GREEK SMALL LETTER ZETA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>231</td>
<td>14/07</td>
<td>GREEK SMALL LETTER ETA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>232</td>
<td>14/08</td>
<td>GREEK SMALL LETTER THETA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>233</td>
<td>14/09</td>
<td>GREEK SMALL LETTER IOTA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>234</td>
<td>14/10</td>
<td>GREEK SMALL LETTER KAPPA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>235</td>
<td>14/11</td>
<td>GREEK SMALL LETTER LAMDA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>236</td>
<td>14/12</td>
<td>GREEK SMALL LETTER MU</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>237</td>
<td>14/13</td>
<td>GREEK SMALL LETTER NU</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>238</td>
<td>14/14</td>
<td>GREEK SMALL LETTER XI</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>239</td>
<td>14/15</td>
<td>GREEK SMALL LETTER OOMICRON</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>240</td>
<td>15/00</td>
<td>GREEK SMALL LETTER PI</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>241</td>
<td>15/01</td>
<td>GREEK SMALL LETTER RHO</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>242</td>
<td>15/02</td>
<td>GREEK SMALL LETTER SIGMA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>243</td>
<td>15/03</td>
<td>GREEK SMALL LETTER FINAL SMALL SIGMA</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>244</td>
<td>15/04</td>
<td>GREEK SMALL LETTER TAU</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>245</td>
<td>15/05</td>
<td>GREEK SMALL LETTER UPSILON</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>246</td>
<td>15/06</td>
<td>GREEK SMALL LETTER PHI</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>247</td>
<td>15/07</td>
<td>GREEK SMALL LETTER CHI</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>248</td>
<td>15/08</td>
<td>GREEK SMALL LETTER PSI</td>
<td>Greek</td>
</tr>
<tr>
<td>007</td>
<td>249</td>
<td>15/09</td>
<td>GREEK SMALL LETTER OMEGA</td>
<td>Greek</td>
</tr>
</tbody>
</table>

<p>| 008   | 161    | 10/01    | LEFT RADICAL                                  | Technical |
| 008   | 162    | 10/02    | TOP LEFT RADICAL                              | Technical |</p>
<table>
<thead>
<tr>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Code Pos</th>
<th>Name</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>163</td>
<td>10,03</td>
<td>HORIZONTAL CONNECTOR</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>164</td>
<td>10,04</td>
<td>TOP INTEGRAL</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>165</td>
<td>10,05</td>
<td>BOTTOM INTEGRAL</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>166</td>
<td>10,06</td>
<td>VERTICAL CONNECTOR</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>167</td>
<td>10,07</td>
<td>TOP LEFT SQUARE BRACKET</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>168</td>
<td>10,08</td>
<td>BOTTOM LEFT SQUARE BRACKET</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>169</td>
<td>10,09</td>
<td>TOP RIGHT SQUARE BRACKET</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>170</td>
<td>10,10</td>
<td>BOTTOM RIGHT SQUARE BRACKET</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>171</td>
<td>10,11</td>
<td>TOP LEFT PARENTHESIS</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>172</td>
<td>10,12</td>
<td>BOTTOM LEFT PARENTHESIS</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>173</td>
<td>10,13</td>
<td>TOP RIGHT PARENTHESIS</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>174</td>
<td>10,14</td>
<td>BOTTOM RIGHT PARENTHESIS</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>175</td>
<td>10,15</td>
<td>LEFT MIDDLE CURLY BRACE</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>176</td>
<td>11,00</td>
<td>RIGHT MIDDLE CURLY BRACE</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>177</td>
<td>11,01</td>
<td>TOP LEFT SUMMATION</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>178</td>
<td>11,02</td>
<td>BOTTOM LEFT SUMMATION</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>179</td>
<td>11,03</td>
<td>TOP VERTICAL SUMMATION CONNECTOR</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>180</td>
<td>11,04</td>
<td>BOTTOM VERTICAL SUMMATION CONNECTOR</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>181</td>
<td>11,05</td>
<td>TOP RIGHT SUMMATION</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>182</td>
<td>11,06</td>
<td>BOTTOM RIGHT SUMMATION</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>183</td>
<td>11,07</td>
<td>RIGHT MIDDLE SUMMATION</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>184</td>
<td>11,12</td>
<td>LESS THAN OR EQUAL SIGN</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>185</td>
<td>11,13</td>
<td>NOT EQUAL SIGN</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>186</td>
<td>11,14</td>
<td>GREATER THAN OR EQUAL SIGN</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>187</td>
<td>11,15</td>
<td>INTEGRAL</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>188</td>
<td>12,00</td>
<td>THEREFORE</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>189</td>
<td>12,01</td>
<td>VARIATION, PROPORTIONAL TO</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>190</td>
<td>12,02</td>
<td>INFINITY</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>191</td>
<td>12,03</td>
<td>NABLA, DEL</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>192</td>
<td>12,04</td>
<td>IS APPROXIMATE TO</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>193</td>
<td>12,05</td>
<td>SIMILAR OR EQUAL TO</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>194</td>
<td>12,06</td>
<td>IF AND ONLY IF</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>195</td>
<td>12,07</td>
<td>IMPLIES</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>196</td>
<td>12,08</td>
<td>IDENTICAL TO</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>197</td>
<td>12,09</td>
<td>RADICAL</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>198</td>
<td>12,10</td>
<td>IS INCLUDED IN</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>199</td>
<td>12,11</td>
<td>INCLUDES</td>
<td>Technical</td>
</tr>
<tr>
<td>008</td>
<td>200</td>
<td>12,12</td>
<td>INTERSECTION</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>201</td>
<td>12,13</td>
<td>UNION</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>202</td>
<td>12,14</td>
<td>LOGICAL AND</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>203</td>
<td>12,15</td>
<td>LOGICAL OR</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>204</td>
<td>12,16</td>
<td>PARTIAL DERIVATIVE</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>205</td>
<td>12,17</td>
<td>FUNCTION</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>206</td>
<td>12,18</td>
<td>LEFT ARROW</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>207</td>
<td>12,19</td>
<td>UPWARD ARROW</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>208</td>
<td>12,20</td>
<td>RIGHT ARROW</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>209</td>
<td>12,21</td>
<td>DOWNWARD ARROW</td>
<td>Technical</td>
</tr>
<tr>
<td>009</td>
<td>210</td>
<td>12,22</td>
<td>Blank</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>211</td>
<td>12,23</td>
<td>SOLID DIAMOND</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>212</td>
<td>12,24</td>
<td>CHECKERBOARD</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>213</td>
<td>12,25</td>
<td>&quot;HT&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>214</td>
<td>12,26</td>
<td>&quot;FF&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>215</td>
<td>12,27</td>
<td>&quot;CR&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>216</td>
<td>12,28</td>
<td>&quot;LF&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>217</td>
<td>12,29</td>
<td>&quot;NL&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>218</td>
<td>12,30</td>
<td>&quot;VT&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>219</td>
<td>12,31</td>
<td>LOWER-RIGHT CORNER</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>220</td>
<td>12,32</td>
<td>UPPER-RIGHT CORNER</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>221</td>
<td>12,33</td>
<td>UPPER-LEFT CORNER</td>
<td>Special</td>
</tr>
</tbody>
</table>

87
<table>
<thead>
<tr>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Code Pos</th>
<th>Name</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>009</td>
<td>237</td>
<td>14/13</td>
<td>LOWER-LEFT CORNER</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>238</td>
<td>14/14</td>
<td>CROSSING-LINES</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>239</td>
<td>14/15</td>
<td>HORIZONTAL LINE, SCAN 1</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>240</td>
<td>15/00</td>
<td>HORIZONTAL LINE, SCAN 3</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>241</td>
<td>15/01</td>
<td>HORIZONTAL LINE, SCAN 5</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>242</td>
<td>15/02</td>
<td>HORIZONTAL LINE, SCAN 7</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>243</td>
<td>15/03</td>
<td>HORIZONTAL LINE, SCAN 9</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>244</td>
<td>15/04</td>
<td>LEFT &quot;T&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>245</td>
<td>15/05</td>
<td>RIGHT &quot;T&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>246</td>
<td>15/06</td>
<td>BOTTOM &quot;T&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>247</td>
<td>15/07</td>
<td>TOP &quot;T&quot;</td>
<td>Special</td>
</tr>
<tr>
<td>009</td>
<td>248</td>
<td>15/08</td>
<td>VERTICAL BAR</td>
<td>Special</td>
</tr>
<tr>
<td>010</td>
<td>161</td>
<td>10/01</td>
<td>EM SPACE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>162</td>
<td>10/02</td>
<td>EN SPACE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>163</td>
<td>10/03</td>
<td>3/EM SPACE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>164</td>
<td>10/04</td>
<td>4/EM SPACE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>165</td>
<td>10/05</td>
<td>DIGIT SPACE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>166</td>
<td>10/06</td>
<td>PUNCTUATION SPACE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>167</td>
<td>10/07</td>
<td>THIN SPACE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>168</td>
<td>10/08</td>
<td>HAIR SPACE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>169</td>
<td>10/09</td>
<td>EM DASH</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>170</td>
<td>10/10</td>
<td>EN DASH</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>172</td>
<td>10/12</td>
<td>SIGNIFICANT BLANK SYMBOL</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>174</td>
<td>10/14</td>
<td>ELLIPSIS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>175</td>
<td>10/15</td>
<td>DOUBLE BASELINE DOT</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>176</td>
<td>11/00</td>
<td>VULGAR FRACTION ONE THIRD</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>177</td>
<td>11/01</td>
<td>VULGAR FRACTION TWO THIRDS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>178</td>
<td>11/02</td>
<td>VULGAR FRACTION ONE FIFTH</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>179</td>
<td>11/03</td>
<td>VULGAR FRACTION TWO FIFTHS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>180</td>
<td>11/04</td>
<td>VULGAR FRACTION THREE FIFTHS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>181</td>
<td>11/05</td>
<td>VULGAR FRACTION FOUR FIFTHS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>182</td>
<td>11/06</td>
<td>VULGAR FRACTION ONE SIXTH</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>183</td>
<td>11/07</td>
<td>VULGAR FRACTION FIVE SIXTHS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>184</td>
<td>11/08</td>
<td>CARE OF</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>187</td>
<td>11/11</td>
<td>FIGURE DASH</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>188</td>
<td>11/12</td>
<td>LEFT ANGLE BRACKET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>189</td>
<td>11/13</td>
<td>DECIMAL POINT</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>190</td>
<td>11/14</td>
<td>RIGHT ANGLE BRACKET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>191</td>
<td>11/15</td>
<td>MARKER</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>195</td>
<td>12/03</td>
<td>VULGAR FRACTION ONE EIGHTH</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>196</td>
<td>12/04</td>
<td>VULGAR FRACTION THREE EIGHTHS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>197</td>
<td>12/05</td>
<td>VULGAR FRACTION FIVE EIGHTHS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>198</td>
<td>12/06</td>
<td>VULGAR FRACTION SEVEN EIGHTHS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>201</td>
<td>12/09</td>
<td>TRADEMARK SIGN</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>202</td>
<td>12/10</td>
<td>SIGNATURE MARK</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>203</td>
<td>12/11</td>
<td>TRADEMARK SIGN IN CIRCLE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>204</td>
<td>12/12</td>
<td>LEFT OPEN TRIANGLE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>205</td>
<td>12/13</td>
<td>RIGHT OPEN TRIANGLE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>206</td>
<td>12/14</td>
<td>EM OPEN CIRCLE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>207</td>
<td>12/15</td>
<td>EM OPEN RECTANGLE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>208</td>
<td>13/00</td>
<td>LEFT SINGLE QUOTATION MARK</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>209</td>
<td>13/01</td>
<td>RIGHT SINGLE QUOTATION MARK</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>210</td>
<td>13/02</td>
<td>LEFT DOUBLE QUOTATION MARK</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>211</td>
<td>13/03</td>
<td>RIGHT DOUBLE QUOTATION MARK</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>212</td>
<td>13/04</td>
<td>PRESCRIPTION, TAKE, RECIPE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>214</td>
<td>13/06</td>
<td>MINUTES</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>215</td>
<td>13/07</td>
<td>SECONDS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>217</td>
<td>13/09</td>
<td>LATIN CROSS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>218</td>
<td>13/10</td>
<td>HEXAGON CROSS</td>
<td>Publish</td>
</tr>
</tbody>
</table>

88
<table>
<thead>
<tr>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Code Pos</th>
<th>Name</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>219</td>
<td>13/11</td>
<td>FILLED RECTANGLE BULLET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>220</td>
<td>13/12</td>
<td>FILLED LEFT TRIANGLE BULLET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>221</td>
<td>13/13</td>
<td>FILLED RIGHT TRIANGLE BULLET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>222</td>
<td>13/14</td>
<td>EM FILLED CIRCLE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>223</td>
<td>13/15</td>
<td>EM FILLED RECTANGLE</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>224</td>
<td>14/00</td>
<td>EN OPEN CIRCLE BULLET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>225</td>
<td>14/01</td>
<td>EN OPEN SQUARE BULLET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>226</td>
<td>14/02</td>
<td>OPEN RECTANGULAR BULLET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>227</td>
<td>14/03</td>
<td>OPEN TRIANGULAR BULLET UP</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>228</td>
<td>14/04</td>
<td>OPEN TRIANGULAR BULLET DOWN</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>229</td>
<td>14/05</td>
<td>OPEN STAR</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>230</td>
<td>14/06</td>
<td>EN FILLED CIRCLE BULLET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>231</td>
<td>14/07</td>
<td>EN FILLED SQUARE BULLET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>232</td>
<td>14/08</td>
<td>FILLED TRIANGULAR BULLET UP</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>233</td>
<td>14/09</td>
<td>FILLED TRIANGULAR BULLET DOWN</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>234</td>
<td>14/10</td>
<td>LEFT POINTER</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>235</td>
<td>14/11</td>
<td>RIGHT POINTER</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>236</td>
<td>14/12</td>
<td>CLUB</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>237</td>
<td>14/13</td>
<td>DIAMOND</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>238</td>
<td>14/14</td>
<td>HEART</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>240</td>
<td>15/00</td>
<td>MALTESE CROSS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>241</td>
<td>15/01</td>
<td>DAGGER</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>242</td>
<td>15/02</td>
<td>DOUBLE DAGGER</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>243</td>
<td>15/03</td>
<td>CHECK MARK, TICK</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>244</td>
<td>15/04</td>
<td>BALLOT CROSS</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>245</td>
<td>15/05</td>
<td>MUSICAL SHARP</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>246</td>
<td>15/06</td>
<td>MUSICAL FLAT</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>247</td>
<td>15/07</td>
<td>MALE SYMBOL</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>248</td>
<td>15/08</td>
<td>FEMALE SYMBOL</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>249</td>
<td>15/09</td>
<td>TELEPHONE SYMBOL</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>250</td>
<td>15/10</td>
<td>TELEPHONE RECORDER SYMBOL</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>251</td>
<td>15/11</td>
<td>PHONOGRAPH COPYRIGHT SIGN</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>252</td>
<td>15/12</td>
<td>CARET</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>253</td>
<td>15/13</td>
<td>SINGLE LOW QUOTATION MARK</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>254</td>
<td>15/14</td>
<td>DOUBLE LOW QUOTATION MARK</td>
<td>Publish</td>
</tr>
<tr>
<td>010</td>
<td>255</td>
<td>15/15</td>
<td>CURSOR</td>
<td>Publish</td>
</tr>
<tr>
<td>011</td>
<td>163</td>
<td>10/03</td>
<td>LEFT CARET</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>166</td>
<td>10/06</td>
<td>RIGHT CARET</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>168</td>
<td>10/08</td>
<td>DOWN CARET</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>169</td>
<td>10/09</td>
<td>UP CARET</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>192</td>
<td>12/00</td>
<td>OVERBAR</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>194</td>
<td>12/02</td>
<td>DOWN TACK</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>195</td>
<td>12/03</td>
<td>UP SHOE (CAP)</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>196</td>
<td>12/04</td>
<td>DOWN STILE</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>198</td>
<td>12/06</td>
<td>UNDERBAR</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>202</td>
<td>12/10</td>
<td>JOT</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>204</td>
<td>12/12</td>
<td>QUAD</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>206</td>
<td>12/14</td>
<td>UP TACK</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>207</td>
<td>12/15</td>
<td>CIRCLE</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>211</td>
<td>13/03</td>
<td>UP STILE</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>214</td>
<td>13/06</td>
<td>DOWN SHOE (CUP)</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>216</td>
<td>13/08</td>
<td>RIGHT SHOE</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>218</td>
<td>13/10</td>
<td>LEFT SHOE</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>220</td>
<td>13/12</td>
<td>LEFT TACK</td>
<td>APL</td>
</tr>
<tr>
<td>011</td>
<td>252</td>
<td>15/12</td>
<td>RIGHT TACK</td>
<td>APL</td>
</tr>
</tbody>
</table>

| 012   | 223   | 13/15    | DOUBLE LOW LINE                   | Hebrew    |
| 012   | 224   | 14/00    | HEBREW LETTER ALEPH               | Hebrew    |

89
<table>
<thead>
<tr>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Code Pos</th>
<th>Name</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>012</td>
<td>225</td>
<td>14/01</td>
<td>HEBREW LETTER BET</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>226</td>
<td>14/02</td>
<td>HEBREW LETTER GIMEL</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>227</td>
<td>14/03</td>
<td>HEBREW LETTER DALET</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>228</td>
<td>14/04</td>
<td>HEBREW LETTER HE</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>229</td>
<td>14/05</td>
<td>HEBREW LETTER WAW</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>230</td>
<td>14/06</td>
<td>HEBREW LETTER ZAIN</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>231</td>
<td>14/07</td>
<td>HEBREW LETTER CHET</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>232</td>
<td>14/08</td>
<td>HEBREW LETTER TET</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>233</td>
<td>14/09</td>
<td>HEBREW LETTER YOD</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>234</td>
<td>14/10</td>
<td>HEBREW LETTER FINAL KAPH</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>235</td>
<td>14/11</td>
<td>HEBREW LETTER KAPHE</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>236</td>
<td>14/12</td>
<td>HEBREW LETTER LAMED</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>237</td>
<td>14/13</td>
<td>HEBREW LETTER FINAL MEM</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>238</td>
<td>14/14</td>
<td>HEBREW LETTER MEM</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>239</td>
<td>14/15</td>
<td>HEBREW LETTER FINAL NUN</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>240</td>
<td>15/00</td>
<td>HEBREW LETTER NUN</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>241</td>
<td>15/01</td>
<td>HEBREW LETTER SAMECH</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>242</td>
<td>15/02</td>
<td>HEBREW LETTER A’YIN</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>243</td>
<td>15/03</td>
<td>HEBREW LETTER FINAL PE</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>244</td>
<td>15/04</td>
<td>HEBREW LETTER PE</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>245</td>
<td>15/05</td>
<td>HEBREW LETTER FINAL ZADE</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>246</td>
<td>15/06</td>
<td>HEBREW LETTER ZADE</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>247</td>
<td>15/07</td>
<td>HEBREW QOPH</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>248</td>
<td>15/08</td>
<td>HEBREW RESH</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>249</td>
<td>15/09</td>
<td>HEBREW SHIN</td>
<td>Hebrew</td>
</tr>
<tr>
<td>012</td>
<td>250</td>
<td>15/10</td>
<td>HEBREW TAW</td>
<td>Hebrew</td>
</tr>
</tbody>
</table>

255 008 00/08 BACKSPACE, BACK SPACE, BACK CHAR Keyboard
255 009 00/09 TAB Keyboard
255 010 00/10 LINEFEED, LF Keyboard
255 011 00/11 CLEAR Keyboard
255 013 00/13 RETURN, ENTER Keyboard
255 019 01/03 PAUSE, HOLD Keyboard
255 020 01/04 SCROLL LOCK Keyboard
255 027 01/11 ESCAPE Keyboard
255 032 02/00 MULTI-KEY CHARACTER PREFACE Keyboard
255 033 02/01 KANJI, KANJI CONVERT Keyboard
255 034 02/02 MUHENKAN Keyboard
255 035 02/03 HENKAN MODE Keyboard
255 036 02/04 ROMAJI Keyboard
255 037 02/05 HIRAGANA Keyboard
255 038 02/06 KATAKANA Keyboard
255 039 02/07 HIRAGANA/KATAKANA TOGGLE Keyboard
255 040 02/08 ZENKAKU Keyboard
255 041 02/09 HANZAKU Keyboard
255 042 02/10 ZENKAKU/HANZAKU TOGGLE Keyboard
255 043 02/11 TOUROKU Keyboard
255 044 02/12 MASSYO Keyboard
255 045 02/13 KANA LOCK Keyboard
255 046 02/14 KANA SHIFT Keyboard
255 047 02/15 EISU SHIFT Keyboard
255 048 03/00 EISU TOGGLE Keyboard
255 080 05/00 HOME Keyboard
255 081 05/01 LEFT, MOVE LEFT, LEFT ARROW Keyboard
255 082 05/02 UP, MOVE UP, UP ARROW Keyboard
255 083 05/03 RIGHT, MOVE RIGHT, RIGHT ARROW Keyboard
255 084 05/04 DOWN, MOVE DOWN, DOWN ARROW Keyboard
255 085 05/05 PRIOR, PREVIOUS Keyboard
255 086 05/06 NEXT Keyboard
255 087 05/07 END, EOL Keyboard
<table>
<thead>
<tr>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Code Pos</th>
<th>Name</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>088</td>
<td>05/08</td>
<td>BEGIN, BOL</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>096</td>
<td>06/00</td>
<td>SELECT, MARK</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>097</td>
<td>06/01</td>
<td>PRINT</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>098</td>
<td>06/02</td>
<td>EXECUTE, RUN, DO</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>099</td>
<td>06/03</td>
<td>INSERT, INSERT HERE</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>101</td>
<td>06/05</td>
<td>UNDO, OOPS</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>102</td>
<td>06/06</td>
<td>REDO, AGAIN</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>103</td>
<td>06/07</td>
<td>MENU</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>104</td>
<td>06/08</td>
<td>FIND, SEARCH</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>105</td>
<td>06/09</td>
<td>CANCEL, STOP, ABORT, EXIT</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>106</td>
<td>06/10</td>
<td>HELP, QUESTION MARK</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>107</td>
<td>06/11</td>
<td>BREAK</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>126</td>
<td>07/14</td>
<td>MODE SWITCH, SCRIPT SWITCH, CHARACTER SET SWITCH</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>127</td>
<td>07/15</td>
<td>NUM LOCK</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>128</td>
<td>08/00</td>
<td>KEYPAD SPACE</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>137</td>
<td>08/09</td>
<td>KEYPAD TAB</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>141</td>
<td>08/13</td>
<td>KEYPAD ENTER</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>145</td>
<td>09/01</td>
<td>KEYPAD F1, PF1, A</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>146</td>
<td>09/02</td>
<td>KEYPAD F2, PF2, B</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>147</td>
<td>09/03</td>
<td>KEYPAD F3, PF3, C</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>148</td>
<td>09/04</td>
<td>KEYPAD F4, PF4, D</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>170</td>
<td>10/10</td>
<td>KEYPAD MULTIPLICATION SIGN, ASTERISK</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>171</td>
<td>10/11</td>
<td>KEYPAD PLUS SIGN</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>172</td>
<td>10/12</td>
<td>KEYPAD SEPARATOR, COMMA</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>173</td>
<td>10/13</td>
<td>KEYPAD MINUS SIGN, HYPHEN</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>174</td>
<td>10/14</td>
<td>KEYPAD DECIMAL POINT, FULL STOP</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>175</td>
<td>10/15</td>
<td>KEYPAD DIVISION SIGN, SOLIDUS</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>176</td>
<td>11/00</td>
<td>KEYPAD DIGIT ZERO</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>177</td>
<td>11/01</td>
<td>KEYPAD DIGIT ONE</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>178</td>
<td>11/02</td>
<td>KEYPAD DIGIT TWO</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>179</td>
<td>11/03</td>
<td>KEYPAD DIGIT THREE</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>180</td>
<td>11/04</td>
<td>KEYPAD DIGIT FOUR</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>181</td>
<td>11/05</td>
<td>KEYPAD DIGIT FIVE</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>182</td>
<td>11/06</td>
<td>KEYPAD DIGIT SIX</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>183</td>
<td>11/07</td>
<td>KEYPAD DIGIT SEVEN</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>184</td>
<td>11/08</td>
<td>KEYPAD DIGIT EIGHT</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>185</td>
<td>11/09</td>
<td>KEYPAD DIGIT NINE</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>189</td>
<td>11/13</td>
<td>KEYPAD EQUALS SIGN</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>190</td>
<td>11/14</td>
<td>F1</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>191</td>
<td>11/15</td>
<td>F2</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>192</td>
<td>12/00</td>
<td>F3</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>193</td>
<td>12/01</td>
<td>F4</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>194</td>
<td>12/02</td>
<td>F5</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>195</td>
<td>12/03</td>
<td>F6</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>196</td>
<td>12/04</td>
<td>F7</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>197</td>
<td>12/05</td>
<td>F8</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>198</td>
<td>12/06</td>
<td>F9</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>199</td>
<td>12/07</td>
<td>F10</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>200</td>
<td>12/08</td>
<td>F11, L1</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>201</td>
<td>12/09</td>
<td>F12, L2</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>202</td>
<td>12/10</td>
<td>F13, L3</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>203</td>
<td>12/11</td>
<td>F14, L4</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>204</td>
<td>12/12</td>
<td>F15, L5</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>205</td>
<td>12/13</td>
<td>F16, L6</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>206</td>
<td>12/14</td>
<td>F17, L7</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>207</td>
<td>12/15</td>
<td>F18, L8</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>208</td>
<td>13/00</td>
<td>F19, L9</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>209</td>
<td>13/01</td>
<td>F20, L10</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>210</td>
<td>13/02</td>
<td>F21, R1</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>211</td>
<td>13/03</td>
<td>F22, R2</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>212</td>
<td>13/04</td>
<td>F23, R3</td>
<td>Keyboard</td>
</tr>
<tr>
<td>Byte 3</td>
<td>Byte 4</td>
<td>Code Pos</td>
<td>Name</td>
<td>Set</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>----------</td>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td>255</td>
<td>213</td>
<td>13/05</td>
<td>F24, R4</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>214</td>
<td>13/06</td>
<td>F25, R5</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>215</td>
<td>13/07</td>
<td>F26, R6</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>216</td>
<td>13/08</td>
<td>F27, R7</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>217</td>
<td>13/09</td>
<td>F28, R8</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>218</td>
<td>13/10</td>
<td>F29, R9</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>219</td>
<td>13/11</td>
<td>F30, R10</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>220</td>
<td>13/12</td>
<td>F31, R11</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>221</td>
<td>13/13</td>
<td>F32, R12</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>222</td>
<td>13/14</td>
<td>F33, R13</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>223</td>
<td>13/15</td>
<td>F34, R14</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>224</td>
<td>14/00</td>
<td>F35, R15</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>225</td>
<td>14/01</td>
<td>LEFT SHIFT</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>226</td>
<td>14/02</td>
<td>RIGHT SHIFT</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>227</td>
<td>14/03</td>
<td>LEFT CONTROL</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>228</td>
<td>14/04</td>
<td>RIGHT CONTROL</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>229</td>
<td>14/05</td>
<td>CAPS LOCK</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>230</td>
<td>14/06</td>
<td>SHIFT LOCK</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>231</td>
<td>14/07</td>
<td>LEFT META</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>232</td>
<td>14/08</td>
<td>RIGHT META</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>233</td>
<td>14/09</td>
<td>LEFT ALT</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>234</td>
<td>14/10</td>
<td>RIGHT ALT</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>235</td>
<td>14/11</td>
<td>LEFT SUPER</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>236</td>
<td>14/12</td>
<td>RIGHT SUPER</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>237</td>
<td>14/13</td>
<td>LEFT HYPER</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>238</td>
<td>14/14</td>
<td>RIGHT HYPER</td>
<td>Keyboard</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>15/15</td>
<td>DELETE, RUBOUT</td>
<td>Keyboard</td>
</tr>
</tbody>
</table>
Appendix B
Protocol Encoding

Syntactic Conventions
All numbers are in decimal, unless prefixed with #x, in which case they are in hexadecimal (base 16).
The general syntax used to describe requests, replies, errors, events, and compound types is:

\[
\text{NameofThing}
\quad \text{encode-form}
\quad \ldots
\quad \text{encode-form}
\]

Each encode-form describes a single component.
For components described in the protocol as:

name: TYPE
the encode-form is:

\[
N \quad \text{TYPE} \quad \text{name}
\]

N is the number of bytes occupied in the data stream, and TYPE is the interpretation of those bytes. For example,

depth: CARD8
becomes:

\[
1 \quad \text{CARD8} \quad \text{depth}
\]

For components with a static numeric value the encode-form is:

\[
N \quad \text{value} \quad \text{name}
\]
The value is always interpreted as an N-byte unsigned integer. For example, the first two bytes of a Window error are always zero (indicating an error in general) and three (indicating the Window error in particular):

\[
1 \quad 0 \quad \text{Error}
1 \quad 3 \quad \text{code}
\]

For components described in the protocol as:

name: \{Name1, Name2\}
the encode-form is:

\[
N \quad \text{name}
\quad \text{value1 Name1}
\quad \ldots
\quad \text{value1 Name1}
\]
The value is always interpreted as an N-byte unsigned integer. Note that the size of N is sometimes larger than that strictly required to encode the values. For example:

\[
\text{class:} \quad \{\text{InputOutput, InputOnly, CopyFromParent}\}
\]
becomes:

\[
2 \quad 0 \quad \text{CopyFromParent}
\]
For components described in the protocol as:

**NAME**: TYPE or **Alternative**...or **Alternative**

the encode-form is:

<table>
<thead>
<tr>
<th>N</th>
<th>TYPE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value</td>
<td>Alternative</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>Alternative</td>
</tr>
<tr>
<td></td>
<td>Alternative</td>
<td></td>
</tr>
</tbody>
</table>

The alternative values are guaranteed not to conflict with the encoding of TYPE. For example:

destination: WINDOW or PointerWindow or InputFocus

becomes:

<table>
<thead>
<tr>
<th>4</th>
<th>WINDOW</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PointerWindow</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>InputFocus</td>
<td></td>
</tr>
</tbody>
</table>

For components described in the protocol as:

**value-mask**: BITMASK

the encode-form is:

<table>
<thead>
<tr>
<th>N</th>
<th>BITMASK</th>
<th>value-mask</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mask1</td>
<td>mask-name1</td>
</tr>
<tr>
<td></td>
<td>mask1</td>
<td>mask-name1</td>
</tr>
</tbody>
</table>

The individual bits in the mask are specified and named, and N is 2 or 4. The most-significant bit in a BITMASK is reserved for use in defining chained (multiword) bitmasks, as extensions augment existing core requests. The precise interpretation of this bit is not yet defined here, although a probable mechanism is that a 1-bit indicates that another N bytes of bitmask follows, with bits within the overall mask still interpreted from least-significant to most-significant with an N-byte unit, with N-byte units interpreted in stream order, and with the overall mask being byte-swapped in individual N-byte units.

For **LIST** of **VALUE** encodings, the request is followed by a section of the form:

VALUES
encode-form
encode-form

listing an encode-form for each VALUE. The **NAME** in each encode-form keys to the corresponding BITMASK bit. The encoding of a VALUE always occupies four bytes, but the number of bytes specified in the encoding-form indicates how many of the least-significant bytes are actually used; the remaining bytes are unused and their values do not matter.

In various cases, the number of bytes occupied by a component will be specified by a lowercase single-letter variable name instead of a specific numeric value, and often some other component will have its value specified as a simple numeric expression involving these variables. Components specified with such expressions are always interpreted as unsigned integers. The scope of such variables is always just the enclosing request, reply, error, event, or compound type structure. For example:

<table>
<thead>
<tr>
<th>2</th>
<th>3+n</th>
<th>request length</th>
</tr>
</thead>
<tbody>
<tr>
<td>4n</td>
<td>LISTofPOINT</td>
<td>points</td>
</tr>
</tbody>
</table>

For unused bytes (the values of the bytes are undefined and do no matter), the encode-form is:

<table>
<thead>
<tr>
<th>N</th>
<th>unused</th>
</tr>
</thead>
</table>
If the number of unused bytes is variable, the encode-form typically is:

\[ p \text{ unused, } p = \text{pad}(E) \]

where \( E \) is some expression, and \( \text{pad}(E) \) is the number of bytes needed to round \( E \) up to a multiple of four.

\[ \text{pad}(E) = (4 - (E \mod 4)) \mod 4 \]

**Common Types**

LIST\text{of}FOO

In this document the LIST\text{of} notation strictly means some number of repetitions of the FOO encoding; the actual length of the list is encoded elsewhere.

SET\text{of}FOO

A set is always represented by a bitmask, with a 1-bit indicating presence in the set.

BITMASK: CARD32
WINDOW: CARD32
PIXMAP: CARD32
CURSOR: CARD32
FONT: CARD32
GCONTEXT: CARD32
COLORMAP: CARD32
DRAWABLE: CARD32
FONTABLE: CARD32
ATOM: CARD32
VISUALID: CARD32
BYTE: 8-bit value
INT8: 8-bit signed integer
INT16: 16-bit signed integer
INT32: 32-bit signed integer
CARD8: 8-bit unsigned integer
CARD16: 16-bit unsigned integer
CARD32: 32-bit unsigned integer
TIMESTAMP: CARD32

**BITGRAVITY**

0	Forget
1	NorthWest
2	North
3	NorthEast
4	West
5	Center
6	East
7	SouthWest
8	South
9	SouthEast
10	Static

**WINGRAVITY**

0	Unmap
1	NorthWest
2	North
3	NorthEast
4	West
5	Center
6	East
7	SouthWest
8  South
9  SouthEast
10 Static

BOOL
0  False
1  True

SETofEVENT
#x00000001  KeyPress
#x00000002  KeyRelease
#x00000004  ButtonPress
#x00000008  ButtonRelease
#x00000010  EnterWindow
#x00000020  LeaveWindow
#x00000040  PointerMotion
#x00000080  PointerMotionHint
#x00000100  Button1Motion
#x00000200  Button2Motion
#x00000400  Button3Motion
#x00000800  Button4Motion
#x00001000  Button5Motion
#x00002000  ButtonMotion
#x00004000  KeymapState
#x00008000  Exposure
#x00100000  VisibilityChange
#x00200000  StructureNotify
#x00400000  ResizeRedirect
#x01000000  SubstructureNotify
#x02000000  SubstructureRedirect
#x04000000  FocusChange
#x08000000  PropertyChange
#x10000000  ColormapChange
#x80000000  OwnerGrabButton
#x00000000  unused but must be zero

SETofPOINTEREVENT
encodings are the same as for SETofEVENT, except with
#xFFFF8003  unused but must be zero

SETofDEVICEEVENT
encodings are the same as for SETofEVENT, except with
#xFFFFF8000  unused but must be zero

KEYSYM: CARD32
KEYCODE: CARD8
BUTTON: CARD8

SETofKEYBUTMASK
#x0001  Shift
#x0002  Lock
#x0004  Control
#x0008  Mod1
#x0010  Mod2
#x0020  Mod3
#x0040  Mod4
#x0080  Mod5
#x0100  Button1
#x0200  Button2
#x0400  Button3
#x0800  Button4
#x1000  Button5
#x0000  unused but must be zero

SETofKEYMASK
encodings are the same as for SETofKEYBUTMASK, except with
#FF00 unused but must be zero

**STRING8: LISTofCARD8**

**STRING16: LISTofCHAR2B**

### CHAR2B

<table>
<thead>
<tr>
<th>CARD8</th>
<th>byte1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARD8</td>
<td>byte2</td>
</tr>
</tbody>
</table>

### POINT

<table>
<thead>
<tr>
<th>INT16</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT16</td>
<td>y</td>
</tr>
</tbody>
</table>

### RECTANGLE

<table>
<thead>
<tr>
<th>INT16</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT16</td>
<td>y</td>
</tr>
<tr>
<td>CARD16</td>
<td>width</td>
</tr>
<tr>
<td>CARD16</td>
<td>height</td>
</tr>
</tbody>
</table>

### ARC

<table>
<thead>
<tr>
<th>INT16</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT16</td>
<td>y</td>
</tr>
<tr>
<td>CARD16</td>
<td>width</td>
</tr>
<tr>
<td>CARD16</td>
<td>height</td>
</tr>
<tr>
<td>INT16</td>
<td>angle1</td>
</tr>
<tr>
<td>INT16</td>
<td>angle2</td>
</tr>
</tbody>
</table>

### HOST

<table>
<thead>
<tr>
<th>family</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>n</td>
</tr>
<tr>
<td>n</td>
</tr>
<tr>
<td>p</td>
</tr>
<tr>
<td>p</td>
</tr>
</tbody>
</table>

### STR

<table>
<thead>
<tr>
<th>STRING8</th>
<th>length of name in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>name</td>
</tr>
</tbody>
</table>

### Errors

#### Request

<table>
<thead>
<tr>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>2 CARD16</td>
</tr>
<tr>
<td>1 CARD8</td>
</tr>
<tr>
<td>21</td>
</tr>
</tbody>
</table>

#### Value

<table>
<thead>
<tr>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>2 CARD16</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>2 CARD16</td>
</tr>
<tr>
<td>1 CARD8</td>
</tr>
<tr>
<td>21</td>
</tr>
</tbody>
</table>

#### Window

<table>
<thead>
<tr>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
2 CARD16 minor opcode
1 CARD8 major opcode
21 unused

Pixmap
1 0 Error
1 4 code
2 CARD16 sequence number
4 CARD32 bad resource id
2 CARD16 minor opcode
1 CARD8 major opcode
21 unused

Atom
1 0 Error
1 5 code
2 CARD16 sequence number
4 CARD32 bad atom id
2 CARD16 minor opcode
1 CARD8 major opcode
21 unused

Cursor
1 0 Error
1 6 code
2 CARD16 sequence number
4 CARD32 bad resource id
2 CARD16 minor opcode
1 CARD8 major opcode
21 unused

Font
1 0 Error
1 7 code
2 CARD16 sequence number
4 CARD32 bad resource id
2 CARD16 minor opcode
1 CARD8 major opcode
21 unused

Match
1 0 Error
1 8 code
2 CARD16 sequence number
4 unused
2 CARD16 minor opcode
1 CARD8 major opcode
21 unused

Drawable
1 0 Error
1 9 code
2 CARD16 sequence number
4 CARD32 bad resource id
2 CARD16 minor opcode
1 CARD8 major opcode
21 unused

Access
1 0 Error
1 10 code
2 CARD16 sequence number
4 unused
2 CARD16 minor opcode
1 CARD8 major opcode
21 unused
<table>
<thead>
<tr>
<th>Alloc</th>
<th>1 0</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 11</td>
<td>code</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CARD8</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>unused</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Colormap</th>
<th>1 0</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 12</td>
<td>code</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>CARD32</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CARD8</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>unused</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GContext</th>
<th>1 0</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 13</td>
<td>code</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>CARD32</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CARD8</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>unused</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDChoice</th>
<th>1 0</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 14</td>
<td>code</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>CARD32</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CARD8</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>unused</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>1 0</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 15</td>
<td>code</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CARD8</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>unused</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Length</th>
<th>1 0</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 16</td>
<td>code</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CARD8</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>unused</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation</th>
<th>1 0</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 17</td>
<td>code</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>unused</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CARD8</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>unused</td>
</tr>
</tbody>
</table>
Keyboards
KEYCODE values are always greater than 7 (and less than 256).
KEYSYM values with the bit #x10000000 set are reserved as vendor-specific.
The names and encodings of the standard KEYSYM values are contained in Appendix A,
Keysym Encoding.

Pointers
BUTTON values are numbered starting with one.

Predefined Atoms

| PRIMARY | 1 | WM_NORMAL_HINTS | 40 |
| SECONDARY | 2 | WM_SIZE_HINTS | 41 |
| ARC | 3 | WM_ZOOM_HINTS | 42 |
| ATOM | 4 | MIN_SPACE | 43 |
| BITMAP | 5 | NORM_SPACE | 44 |
| CARDINAL | 6 | MAX_SPACE | 45 |
| COLOMAP | 7 | END_SPACE | 46 |
| CURSOR | 8 | SUPERSCRIPT_X | 47 |
| CUT_BUFFER0 | 9 | SUPERSCRIPT_Y | 48 |
| CUT_BUFFER1 | 10 | SUBSCRIPT_X | 49 |
| CUT_BUFFER2 | 11 | SUBSCRIPT_Y | 50 |
| CUT_BUFFER3 | 12 | UNDERLINE_POSITION | 51 |
| CUT_BUFFER4 | 13 | UNDERLINE_THICKNESS | 52 |
| CUT_BUFFER5 | 14 | STIKEOUT_ASCENT | 53 |
| CUT_BUFFER6 | 15 | STIKEOUT_DESCENT | 54 |
| CUT_BUFFER7 | 16 | ITALIC_ANGLE | 55 |
| DRAWABLE | 17 | X_HEIGHT | 56 |
| FONT | 18 | QUAD_WIDTH | 57 |
| INTEGER | 19 | WEIGHT | 58 |
| PIXMAP | 20 | POINT_SIZE | 59 |
| POINT | 21 | RESOLUTION | 60 |
| RECTANGLE | 22 | COPYRIGHT | 61 |
| RESOURCE_MANAGER | 23 | NOTICE | 62 |
| RGB_COLOR_MAP | 24 | FONT_NAME | 63 |
| RGB_BLUE_MAP | 25 | FAMILY_NAME | 64 |
| RGB_DEFAULT_MAP | 26 | FULL_NAME | 65 |
| RGB_GRAY_MAP | 27 | CAP_HEIGHT | 66 |
| RGB_RED_MAP | 28 | WM_CLASS | 67 |
| RGB_GREEN_MAP | 29 | WM_TRANSIENT_FOR | 68 |
| STRING | 30 | WM_CLIENT_MACHINE | 36 |
| VISUALID | 31 | WM_ICON_NAME | 37 |
| WINDOW | 32 | WM_ICON_SIZE | 38 |
| WM_COMMAND | 33 | WM_NAME | 39 |
| WM_HINTS | 34 | WM_CLIENT_MACHINE | 36 |
| WM_CLIENT_MACHINE | 35 | WM_ICON_NAME | 37 |
| WM_ICON_SIZE | 36 | WM_NAME | 39 |

Connection Setup
For TCP connections, displays on a given host are numbered starting from 0, and the server
for display N listens and accepts connections on port 6000 + N. For DECnet connections,
 displays on a given host are numbered starting from 0, and the server for display N listens and
accepts connections on the object name obtained by concatenating "X$X" with the decimal
representation of N, for example, X$X0 and X$X1.
Information sent by the client at connection setup:

1  #x42 MSB first byte-order
1  #x6C LSB first
2  CARD16 protocol-major-version
X Protocol

Xll, Release 5

2 CARD16 protocol-minor-version
2 n length of authorization-protocol-name
2 d length of authorization-protocol-data
2 unused
n STRING8 authorization-protocol-name
p unused, p=pad(n)
d STRING8 authorization-protocol-data
q unused, q=pad(d)

Except where explicitly noted in the protocol, all 16-bit and 32-bit quantities sent by the client must be transmitted with the specified byte order, and all 16-bit and 32-bit quantities returned by the server will be transmitted with this byte order.

Information received by the client if authorization fails:
1 0 failed
1 n length of reason in bytes
2 CARD16 protocol-major-version
2 CARD16 protocol-minor-version
2 (n+p)/4 length in 4-byte units of "additional data"
n STRING8 reason
p unused, p=pad(n)

Information received by the client if authorization is accepted:
1 1 success
1 unused
2 CARD16 protocol-major-version
2 CARD16 protocol-minor-version
2 8+2(n+v+p+m)/4 length in 4-byte units of "additional data"
4 CARD32 release-number
4 CARD32 resource-id-base
4 CARD32 resource-id-mask
4 CARD32 motion-buffer-size
2 v length of vendor
2 CARD16 maximum-request-length
1 CARD8 number of SCREENs in roots
1 n number for FORMATs in pixmap-formats
1 image-byte-order
0 LSBFirst
1 MSBFirst
1 0 LeastSignificant
1 MostSignificant
1 CARD8 bitmap-format-scanline-unit
1 CARD8 bitmap-format-scanline-pad
1 KEYCODE min-keycode
1 KEYCODE max-keycode
4 unused
v STRING8 vendor
p unused, p=pad(v)
8n LISTofFORMAT pixmap-formats
m LISTofSCREEN roots (m is always a multiple of 4)

FORMAT
1 CARD8 depth
1 CARD8 bits-per-pixel
1 CARD8 scanline-pad
5 unused

SCREEN
4 WINDOW root
4 COLOORMAP default-colormap
4 CARD32 white-pixel
4 CARD32 black-pixel
4 SETofEVENT current-input-masks
2 CARD16 width-in-pixels
X Protocol

X11, Release 5

2 CARD16
height-in-pixels
2 CARD16
width-in-millimeters
2 CARD16
height-in-millimeters
2 CARD16
min-installed-maps
2 CARD16
max-installed-maps
4 VISIBLE
root-visual
1
backing-stores
0
Never
1
WhenMapped
2
Always
1 BOOL
save-unders
1 CARD8
root-depth
1 CARD8
number of DEPTHS in allowed-depths
n LISTofDEPTH
allowed-depths (n is always a multiple of 4)

DEPTCH
1 CARD8
depth
1 unused
2 n
number of VISUALTYPES in visuals
4 unused
24n LISTofVISUALTYPE
visuals

VISUALTYPE
4 VISIBLE
visual-id
class
1
0 StaticGray
1 Grayscale
2 StaticColor
3 PseudoColor
4 TrueColor
5 DirectColor
1 CARD8
bits-per-rgb-value
2 CARD16
colormap-entries
4 CARD32
red-mask
4 CARD32
green-mask
4 CARD32
blue-mask
4 unused

Requests
CreateWindow
1
1
opcode
1 CARD8
depth
2 8+n
request length
4 WINDOW
wid
4 WINDOW
parent
4 INT16
x
4 INT16
y
2 CARD16
width
2 CARD16
height
2 CARD16
border-width
2 CARD16
class
0
CopyFromParent
1
InputOutput
2
InputOnly
4 VISIBLE
visual
0
CopyFromParent
4 BITMASK
value-mask (has n bits set to 1)
#x00000001 background-pixmap
#x00000002 background-pixel
#x00000004 border-pixmap
#x00000008 border-pixel
#x00000010 bit-gravity
#x00000020 win-gravity
#x00000040 backing-store
X Protocol

XI1, Release 5

#x00000080 backing-planes
#x00000100 backing-pixel
#x00000200 override-redirect
#x00000400 save-under
#x00000800 event-mask
#x00001000 do-not-propagate-mask
#x00002000 colormap
#x00004000 cursor

4n LISTofVALUE value-list

VALUES

4 PIXMAP background-pixmap
0 None
1 ParentRelative

4 CARD32 background-pixel
4 PIXMAP border-pixmap
0 CopyFromParent
4 CARD32 border-pixel
1 BITGRAVITY bit-gravity
1 WINGRAVITY win-gravity
1 backing-store
0 NotUseful
1 WhenMapped
2 Always

4 CARD32 backing-planes
4 CARD32 backing-pixel
1 BOOL override-redirect
1 BOOL save-under
4 SETofEVENT event-mask
4 SETofDEVECEEVENT do-not-propagate-mask
4 COLORMAP colormap
0 CopyFromParent

4 CURSOR cursor
0 None

ChangeWindowAttributes

1 2 opcode
1 unused
2 3+n request length
4 WINDOW window
4 BITMASK value-mask (has n bits set to 1)
encodings are the same as for CreateWindow

4n LISTofVALUE value-list
encodings are the same as for CreateWindow

GetWindowAttributes

1 3 opcode
1 unused
2 request length
4 WINDOW window

=>

1 1 Reply
1 backing-store
0 NotUseful
1 WhenMapped
2 Always
2 CARD16 sequence number
4 3 reply length
4 VISUALID visual
2 class
1 InputOutput
2 InputOnly
1 BITGRAVITY bit-gravity
1 WINGRAVITY win-gravity
4 CARD32 backing-planes

103
4 CARD32  
1 BOOL  
1 BOOL  
1 0 Unmapped  
1 Unviewable  
2 Viewable  
1 BOOL  
4 COLOMAP  
0 None  
4 SETofEVENT  
4 SETofEVENT  
2 SETofDEVICEEVENT  
4 COLORMAP  

DestroyWindow  
1 4 opcode  
1 unused  
2 2 request length  
4 WINDOW  

DestroySubwindows  
1 5 opcode  
1 unused  
2 2 request length  
4 WINDOW  

ChangeSaveSet  
1 6 opcode  
1 mode  
0 Insert  
1 Delete  
2 2 request length  
4 WINDOW  

ReparentWindow  
1 7 opcode  
1 unused  
2 4 request length  
4 WINDOW  
4 WINDOW  
2 INT16  
2 INT16  

MapWindow  
1 8 opcode  
1 unused  
2 2 request length  
4 WINDOW  

MapSubwindows  
1 9 opcode  
1 unused  
2 2 request length  
4 WINDOW  

UnmapWindow  
1 10 opcode  
1 unused  
2 2 request length  
4 WINDOW  

UnmapSubwindows  
1 11 opcode  
1 unused  
2 2 request length
4 WINDOW

ConfigureWindow
1 12
1 opcode
2 3+n request length
4 WINDOW window
2 BITMASK

#x0001 x
#x0002 y
#x0004 width
#x0008 height
#x0010 border-width
#x0020 sibling
#x0040 stack-mode

2 unused
4n LISTofVALUE value-list

VALUEs
2 INT16 x
2 INT16 y
2 CARD16 width
2 CARD16 height
2 CARD16 border-width
4 WINDOW sibling

1
0 Above
1 Below
2 Toplf
3 Bottomlf
4 Opposite

CirculateWindow
1 13 opcode
1 0 RaiseLowest direction
1 LowerHighest
2 2 request length
4 WINDOW window

GetGeometry
1 14 opcode
1 unused
2 2 request length
4 DRAWABLE drawable

=>
1 1 Reply
1 CARD8 depth
2 CARD16 sequence number
4 0 reply length
4 WINDOW root
2 INT16 x
2 INT16 y
2 CARD16 width
2 CARD16 height
2 CARD16 border-width
10 unused

QueryTree
1 15 opcode
1 unused
2 2 request length
4 WINDOW window

=>
1 1 Reply
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>unused</td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td>4</td>
<td>n</td>
</tr>
<tr>
<td>4</td>
<td>WINDOW</td>
</tr>
<tr>
<td></td>
<td>WINDOW</td>
</tr>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>14</td>
<td>unused</td>
</tr>
</tbody>
</table>

### InternAtom

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>BOOL</td>
</tr>
<tr>
<td>2</td>
<td>2+(n+p)/4</td>
</tr>
<tr>
<td>2</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>unused</td>
</tr>
<tr>
<td>n</td>
<td>STRING8</td>
</tr>
<tr>
<td>p</td>
<td>unused, p=pad(n)</td>
</tr>
</tbody>
</table>

### GetAtomName

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>unused</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>ATOM</td>
</tr>
</tbody>
</table>

### ChangeProperty

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>mode</td>
</tr>
<tr>
<td>0</td>
<td>Replace</td>
</tr>
<tr>
<td>1</td>
<td>Prepend</td>
</tr>
<tr>
<td>2</td>
<td>Append</td>
</tr>
<tr>
<td>2</td>
<td>6+(n+p)/4</td>
</tr>
<tr>
<td>4</td>
<td>WINDOW</td>
</tr>
<tr>
<td>4</td>
<td>ATOM</td>
</tr>
<tr>
<td>4</td>
<td>ATOM</td>
</tr>
<tr>
<td>1</td>
<td>CARD8</td>
</tr>
<tr>
<td>3</td>
<td>unused</td>
</tr>
<tr>
<td>4</td>
<td>CARD32</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>LISTofBYTE</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>unused</td>
</tr>
</tbody>
</table>
DeleteProperty
1 19  opcode
1 unused
2 3  request length
4 WINDOW  window
4 ATOM  property

GetProperty
1 20  opcode
1 BOOL  delete
2 6  request length
4 WINDOW  window
4 ATOM  property
4 ATOM  type
0  AnyPropertyType
4 CARD32  long-offset
4 CARD32  long-length

=>
1 1  Reply
1 CARD8  format
2 CARD16  sequence number
4 \((n+p)/4\)  reply length
4 ATOM  type
0  None
4 CARD32  bytes-after
4 CARD32  length of value in format units
(\(= 0\) for format = 0)
(\(= n\) for format = 8)
(\(= n/2\) for format = 16)
(\(= n/4\) for format = 32)
12 unused
n LISTofBYTE  value
(n is zero for format = 0)
(n is a multiple of 2 for format = 16)
(n is a multiple of 4 for format = 32)
unused, \(p=\text{pad}(n)\)

ListProperties
1 21  opcode
1 unused
2 2  request length
4 WINDOW  window

=>
1 1  Reply
1 unused
2 CARD16  sequence number
4 n  reply length
2 n  number of ATOMs in atoms
22 unused
4n LISTofATOM  atoms

SetSelectionOwner
1 22  opcode
1 unused
2 4  request length
4 WINDOW  owner
0  None
4 ATOM  selection
4 TIMESTAMP  time
0  CurrentTime

GetSelectionOwner
1 23  opcode
1 unused
X Protocol

Xll, Release 5

2 2 request length
4 ATOM selection

=>
1 1 Reply
1 unused
2 CARD16 sequence number
4 0 reply length
4 WINDOW owner
0 None unused

ConvertSelection
1 24 opcode
1 unused
2 6 request length
4 WINDOW requestor
4 ATOM selection
4 ATOM target
4 ATOM property
0 None time
4 TIMESTAMP None CurrentTime

SendEvent
1 25 opcode
1 BOOL propagate
2 11 request length
4 WINDOW destination
0 PointerWindow
1 InputFocus
4 SETofEVENT event-mask
32 event
standard event format (see the Events section)

GrabPointer
1 26 opcode
1 BOOL owner-events
2 6 request length
4 WINDOW grab-window
2 SETofPOINTEREVENT event-mask
1 pointer-mode
0 Synchronous
1 Asynchronous
1 0 Synchronous
1 Asynchronous
4 WINDOW None confine-to
4 CURSOR None cursor
4 TIMESTAMP None time

=>
1 1 Reply
1 status
0 Success
1 AlreadyGrabbed
2 InvalidTime
3 NotViewable
4 Frozen
2 CARD16 sequence number
4 0 reply length
24 unused
UngrabPointer
1 27 opcode
1
2 2 unused
4 TIMESTAMP request length
   0 CurrentTime

GrabButton
1 28 opcode
1 BOOL owner-events
2 6 request length
4 WINDOW grab-window
2 SETOfPOINTEREVENT event-mask
1
   0 Synchronous pointer-mode
   1 Asynchronous
1
   0 Synchronous keyboard-mode
   1 Asynchronous
4 WINDOW request length
   0 None confine-to
   4 CURSOR None cursor
   0 BUTTON None button
   1
1
2 SETOfKEYMASK request length
   #x8000 modifiers
   AnyModifier

UngrabButton
1 29 opcode
1 BUTTON request length
   0 AnyButton
2 3 button
4 WINDOW request length
   0 None
   4 CURSOR None
   0 BUTTON None
   1
1
2 SETOfKEYMASK request length
   #x8000 modifiers
   AnyModifier

ChangeActivePointerGrab
1 30 opcode
1 unused
2 4 request length
4 CURSOR request length
   0 None
4 TIMESTAMP request length
   0 CurrentTime
2 SETOfPOINTEREVENT request length
   event-mask
   unused

GrabKeyboard
1 31 opcode
1 BOOL owner-events
2 4 request length
4 WINDOW grab-window
4 TIMESTAMP request length
   0 CurrentTime
   0 Synchronous pointer-mode
   1 Asynchronous
1
   0 Synchronous keyboard-mode
   1 Asynchronous
2 unused

=>
X Protocol

1 1 Reply status
0 Success
1 AlreadyGrabbed
2 InvalidTime
3 NotViewable
4 Frozen

2 CARD16 sequence number
4 0 reply length
24 unused

UngrabKeyboard
1 32 opcode
1 unused
2 2 request length
4 TIME STAMP time
0 CurrentTime

GrabKey
1 33 opcode
1 BOOL owner-events
2 4 request length
4 WINDOW grab-window
2 SETofKEYMASK modifiers
#x8000 AnyModifier
1 KEYCODE key
0 AnyKey
1 Synchronous
0 Asynchronous
1 Synchronous
1 Asynchronous
3 unused

UngrabKey
1 34 opcode
1 KEYCODE key
0 AnyKey
2 3 request length
4 WINDOW grab-window
2 SETofKEYMASK modifiers
#x8000 AnyModifier
2 unused

AllowEvents
1 35 opcode
1 mode
0 AsyncPointer
1 SyncPointer
2 ReplayPointer
3 AsyncKeyboard
4 SyncKeyboard
5 ReplayKeyboard
6 AsyncBoth
7 SyncBoth
2 2 request length
4 TIMESTAMP time
0 CurrentTime

GrabServer
1 36 opcode
1 unused
2 1 request length
UngrabServer
1 37  opcode
1 unused
2 1  request length

QueryPointer
1 38  opcode
1 unused
2 2  request length
4 WINDOW  window

=>
1 1  Reply
1 BOOL  same-screen
2 CARD16  sequence number
4 0  reply length
4 WINDOW  root
4 WINDOW  child
0 None
2 INT16  root-x
2 INT16  root-y
2 INT16  win-x
2 INT16  win-y
2 SETofKEYBUTMASK  mask
6 unused

GetMotionEvents
1 39  opcode
1 unused
2 4  request length
4 WINDOW  window
4 TIMESTAMP  start
0 CurrentTime
4 TIMESTAMP  stop
0 CurrentTime

=>
1 1  Reply
1 unused
2 CARD16  sequence number
4 2n  reply length
4 n  number of TIMECOORDs in events
20 unused
8n LISTofTIMECOORD  events

TIMECOORD
4 TIMESTAMP  time
2 INT16  x
2 INT16  y

TranslateCoordinates
1 40  opcode
1 unused
2 4  request length
4 WINDOW  src-window
4 WINDOW  dst-window
2 INT16  src-x
2 INT16  src-y

=>
1 1  Reply
1 BOOL  same-screen
2 CARD16  sequence number
4 0  reply length
4 WINDOW  child
0 None
2 INT16  dst-x
2  INT16  dst-y
16  unused

WarpPointer
1  41  opcode
1  unused
2  6  request length
4  WINDOW  src-window
0  None
4  WINDOW  dst-window
0  None
2  INT16  src-x
2  INT16  src-y
2  CARD16  src-width
2  CARD16  src-height
2  INT16  dst-x
2  INT16  dst-y

SetInputFocus
1  42  opcode
1  revert-to
0  None
1  PointerRoot
2  Parent
2  3  request length
4  WINDOW  focus
0  None
1  PointerRoot
4  TIMESTAMP  time
0  CurrentTime

GetInputFocus
1  43  opcode
1  unused
2  1  request length
=>
1  1  Reply
1  revert-to
0  None
1  PointerRoot
2  Parent
2  CARD16  sequence number
4  0  reply length
4  WINDOW  focus
0  None
1  PointerRoot
20  unused

QueryKeymap
1  44  opcode
1  unused
2  1  request length
=>
1  1  Reply
1  unused
2  CARD16  sequence number
4  2  reply length
32 LISTofCARD8  keys

OpenFont
1  45  opcode
1  unused
2  3+(n+p)/4  request length
4  FONT  fid
X Protocol

2 n
2 n STRING8
p

CloseFont
1 46
1 unused
2 2
request length
4 FONT

QueryFont
1 47
1 unused
2 2
request length
4 FONTABLE

=>
1 1
Reply
1 unused
2 CARD16
sequence number
4 7+2n+3m
reply length
12 CHARINFO
min-bounds
4 unused
12 CHARINFO
max-bounds
4 unused
2 CARD16
min-char-or-byte2
2 CARD16
max-char-or-byte2
2 CARD16
default-char
2 n
number of FONTPROPs in properties
1 draw-direction
0 LeftToRight
1 RightToLeft
1 CARD8
min-byte1
1 CARD8
max-byte1
1 BOOL
all-chars-exist
2 INT16
font-ascent
2 INT16
font-descent
4 m
number of CHARINFOs in char-infos
8n LISTofFONTPROP
properties
12m LISTofCHARINFO
char-infos

FONTPROP
4 ATOM
name
4 <32-bits>
value

CHARINFO
2 INT16
left-side-bearing
2 INT16
right-side-bearing
2 INT16
character-width
2 INT16
ascent
2 INT16
descent
2 CARD16
attributes

QueryTextExtents
1 48
opcode
1 BOOL
odd length, True if p = 2
2 2+(2n+p)/4
request length
4 FONTABLE
font
2n STRING16
string
p
unused, p=pad(2n)

=>
1 1
Reply
1 draw-direction
0 LeftToRight
1 RightToLeft
<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>CARD16</td>
<td>sequence number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>reply length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
<td>font-ascent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
<td>font-descent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
<td>overall-ascent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
<td>overall-descent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INT32</td>
<td>overall-width</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INT32</td>
<td>overall-left</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INT32</td>
<td>overall-right</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ListFonts**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49</td>
<td>opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2+(n+p)/4</td>
<td>request length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
<td>max-names</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>STRING8</td>
<td>pattern</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td></td>
<td>unused, p=pad(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

=>

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Reply</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
<td>sequence number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(n+p)/4</td>
<td>reply length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
<td>number of STRs in names</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>LISTofSTR</td>
<td>names</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td></td>
<td>unused, p=pad(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ListFontsWithInfo**

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2+(n+p)/4</td>
<td>request length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
<td>max-names</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>STRING8</td>
<td>pattern</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td></td>
<td>unused, p=pad(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

=> (except for last in series)

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Reply</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
<td>length of name in bytes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
<td>sequence number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7+2m+(n+p)/4</td>
<td>reply length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>CHARINFO</td>
<td>min-bounds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>CHARINFO</td>
<td>max-bounds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
<td>min-char-or-byte2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
<td>max-char-or-byte2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
<td>default-char</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td></td>
<td>number of FONTPROPs in properties</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>LeftToRight</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>RightToLeft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>CARD8</td>
<td>min-byte1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>CARD8</td>
<td>max-byte1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>BOOL</td>
<td>all-chars-exist</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
<td>font-ascent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
<td>font-descent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CARD32</td>
<td>replies-hint</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8m</td>
<td>LISTofFONTPROP</td>
<td>properties</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>STRING8</td>
<td>name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td></td>
<td>unused, p=pad(n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FONTPROP**

encodings are the same as for QueryFont
CHARINFO
  encodings are the same as for QueryFont
=> (last in series)
  1  1  Reply
  1  0  last-reply indicator
  2  CARD16  sequence number
  4  7  reply length
  52  unused

SetFontPath
  1  51  opcode
  1  unused
  2  2*(n+p)/4  request length
  2  CARD16  number of STRs in path
  2  unused
  n  LISTofSTR  path
  p  unused, p=pad(n)

GetFontPath
  1  52  opcode
  1  unused
  2  1  request list
=>
  1  1  Reply
  1  unused
  2  CARD16  sequence number
  4  (n+p)/4  reply length
  2  CARD16  number of STRs in path
  22  unused
  n  LISTofSTR  path
  p  unused, p=pad(n)

CreatePixmap
  1  53  opcode
  1  CARD8  depth
  2  4  request length
  4  PIXMAP  pid
  4  DRAWABLE  drawable
  2  CARD16  width
  2  CARD16  height

FreePixmap
  1  54  opcode
  1  unused
  2  2  request length
  4  PIXMAP  pixmap

CreateGC
  1  55  opcode
  1  unused
  2  4+n  request length
  4  GCONTEXT  cid
  4  DRAWABLE  drawable
  4  BITMASK  value-mask (has n bits set to 1)
  #x00000001  function
  #x00000002  plane-mask
  #x00000004  foreground
  #x00000008  background
  #x00000010  line-width
  #x00000020  line-style
  #x00000040  cap-style
  #x00000080  join-style
  #x00000100  fill-style
  #x00000200  fill-rule
X Protocol

4n LISTofVALUE VALUEs

VALUES
definitions

0 Clear
1 And
2 AndReverse
3 Copy
4 AndInverted
5 NoOp
6 Xor
7 Or
8 Nor
9 Equiv
10 Invert
11 OrReverse
12 CopyInverted
13 OrInverted
14 Nand
15 Set

4 CARD32 plane-mask
4 CARD32 foreground
4 CARD32 background
2 CARD16 line-width
1 line-style

0 Solid
1 OnOffDash
2 DoubleDash
1 cap-style

0 NotLast
1 Butt
2 Round
3 Projecting
1 join-style

0 Miter
1 Round
2 Bevel
1 fill-style

0 Solid
1 Tiled
2 Stippled
3 OpaqueStippled
1 fill-rule

0 EvenOdd
1 Winding

4 PIXMAP tile
4 PIXMAP stipple
2 INT16 tile-stipple-x-origin
2 INT16 tile-stipple-y-origin
4 FONT font
1 subwindow-mode

0 ClipByChildren
1 IncludeInferiors
1 BOOL graphics-exposures
2 INT16 clip-x-origin
2 INT16 clip-y-origin
4 PIXMAP clip-mask
 0 None
2 CARD16 dash-offset
1 CARD8 dashes
1
 0 Chord
 1 PieSlice

ChangeGC
1 56 opcode
1 unused
2 3+n request length
4 GCONTEXT gc
4 BITMASK value-mask (has n bits set to 1)
 4n LISTofVALUE value-list
encodings are the same as for CreateGC

CopyGC
1 57 opcode
1 unused
2 4 request length
4 GCONTEXT src-gc
4 GCONTEXT dst-gc
4 BITMASK value-mask
encodings are the same as for CreateGC

SetDashes
1 58 opcode
1 unused
2 3+(n+p)/4 request length
4 GCONTEXT gc
2 CARD16 dash-offset
2 n length of dashes
n LISTofCARD8 dashes
p unused, p=pad(n)

SetClipRectangles
1 59 opcode
1 ordering
 0 UnSorted
 1 YSorted
 2 YXSorted
 3 YX Banded
2 3+2n request length
4 GCONTEXT gc
2 INT16 clip-x-origin
2 INT16 clip-y-origin
8n LISTofRECTANGLE rectangles

FreeGC
1 60 opcode
1 unused
2 2 request length
4 GCONTEXT gc

ClearArea
1 61 opcode
1 BOOL exposures
2 4 request length
4 WINDOW window
2 INT16 x
2 INT16 y

117
2 CARD16          width
2 CARD16          height

**CopyArea**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>62</td>
</tr>
<tr>
<td>1</td>
<td>unused</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>DRAWABLE</td>
</tr>
<tr>
<td>4</td>
<td>DRAWABLE</td>
</tr>
<tr>
<td>4</td>
<td>GCONTEXT</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
</tr>
</tbody>
</table>

**CopyPlane**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>63</td>
</tr>
<tr>
<td>1</td>
<td>unused</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>DRAWABLE</td>
</tr>
<tr>
<td>4</td>
<td>DRAWABLE</td>
</tr>
<tr>
<td>4</td>
<td>GCONTEXT</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td>2</td>
<td>CARD16</td>
</tr>
<tr>
<td>4</td>
<td>CARD32</td>
</tr>
</tbody>
</table>

**PolyPoint**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>Origin</td>
</tr>
<tr>
<td>0</td>
<td>Previous</td>
</tr>
<tr>
<td>2</td>
<td>3+n</td>
</tr>
<tr>
<td>4</td>
<td>DRAWABLE</td>
</tr>
<tr>
<td>4</td>
<td>GCONTEXT</td>
</tr>
<tr>
<td>4n</td>
<td>LISTofPOINT</td>
</tr>
</tbody>
</table>

**PolyLine**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>65</td>
</tr>
<tr>
<td>0</td>
<td>Origin</td>
</tr>
<tr>
<td>0</td>
<td>Previous</td>
</tr>
<tr>
<td>2</td>
<td>3+n</td>
</tr>
<tr>
<td>4</td>
<td>DRAWABLE</td>
</tr>
<tr>
<td>4</td>
<td>GCONTEXT</td>
</tr>
<tr>
<td>4n</td>
<td>LISTofPOINT</td>
</tr>
</tbody>
</table>

**PolySegment**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>66</td>
</tr>
<tr>
<td>1</td>
<td>unused</td>
</tr>
<tr>
<td>2</td>
<td>3+2n</td>
</tr>
<tr>
<td>4</td>
<td>DRAWABLE</td>
</tr>
<tr>
<td>4</td>
<td>GCONTEXT</td>
</tr>
<tr>
<td>8n</td>
<td>LISTofSEGMENT</td>
</tr>
</tbody>
</table>

**SEGMENT**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
<tr>
<td>2</td>
<td>INT16</td>
</tr>
</tbody>
</table>
PolyRectangle
1  67  opcode
1  unused
2 3+2n  request length
4  DRAWABLE  drawable
4  GCONTEXT  gc
8n LISTofRECTANGLE  rectangles

PolyArc
1  68  opcode
1  unused
2 3+3n  request length
4  DRAWABLE  drawable
4  GCONTEXT  gc
12n LISTofARC  arcs

FillPoly
1  69  opcode
1  unused
2 4+n  request length
4  DRAWABLE  drawable
4  GCONTEXT  gc
1
0  Complex
1  Nonconvex
2  Convex
1
0  Origin
1  Previous
2  unused
4n LISTofPOINT  points

PolyFillRectangle
1  70  opcode
1  unused
2 3+2n  request length
4  DRAWABLE  drawable
4  GCONTEXT  gc
8n LISTofRECTANGLE  rectangles

PolyFillArc
1  71  opcode
1  unused
2 3+3n  request length
4  DRAWABLE  drawable
4  GCONTEXT  gc
12n LISTofARC  arcs

PutImage
1  72  opcode
1
0  Bitmap
1  XYPixmap
2  ZPixmap
2 6+(n+p)/4  request length
4  DRAWABLE  drawable
4  GCONTEXT  gc
2  CARD16  width
2  CARD16  height
2  INT16  dst-x
2  INT16  dst-y
1  CARD8  left-pad
1  CARD8  depth
2  unused
n LISTofBYTE  data
p  unused, p=pad(n)
### GetImage

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Format</th>
<th>Request Length</th>
<th>Drawable</th>
<th>X</th>
<th>Y</th>
<th>Width</th>
<th>Height</th>
<th>Plane-Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>XYPixmap</td>
<td>5</td>
<td>1 INT16</td>
<td>2 INT16</td>
<td>2 CARD16</td>
<td>4 CARD16</td>
<td>4 CARD32</td>
<td></td>
</tr>
</tbody>
</table>

### Reply

<table>
<thead>
<tr>
<th>Depth</th>
<th>Sequence Number</th>
<th>Reply Length</th>
<th>Visual</th>
<th>Unused</th>
<th>Data</th>
<th>Unused, p=pad(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARD8</td>
<td>(n+p)/4</td>
<td>0 None</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### PolyText8

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Unused</th>
<th>Request Length</th>
<th>Drawable</th>
<th>GC</th>
<th>X</th>
<th>Y</th>
<th>Items</th>
<th>Unused, p=pad(n) (p is always 0 or 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>74</td>
<td></td>
<td>4+(n+p)/4</td>
<td>1 INT16</td>
<td>2</td>
<td>2 INT16</td>
<td>n LISTofTEXTITEM8</td>
<td>p TEXTITEM8</td>
<td>length of string (cannot be 255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>or 255</td>
<td>font-shift indicator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 INT8</td>
<td>font byte 3 (most-significant)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2m STRING8</td>
<td>font byte 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>font byte 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>font byte 0 (least-significant)</td>
</tr>
</tbody>
</table>

### PolyText16

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Unused</th>
<th>Request Length</th>
<th>Drawable</th>
<th>GC</th>
<th>X</th>
<th>Y</th>
<th>Items</th>
<th>Unused, p=pad(n) (p must be 0 or 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td></td>
<td>4+(n+p)/4</td>
<td>1 INT16</td>
<td>2</td>
<td>2 INT16</td>
<td>n LISTofTEXTITEM16</td>
<td>p TEXTITEM16</td>
<td>number of CHAR2Bs in string (cannot be 255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>or 255</td>
<td>font-shift indicator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 INT8</td>
<td>font byte 3 (most-significant)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2m STRING16</td>
<td>font byte 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>font byte 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>font byte 0 (least-significant)</td>
</tr>
</tbody>
</table>
ImageText8
1 76
1 n
2 4+(n+p)/4
4 DRAWABLE
drawable
4 GCONTEXT
gc
2 INT16
x
2 INT16
y
n STRING8
string
p unused, p=pad(n)

ImageText16
1 77
1 n
2 4+(2n+p)/4
4 DRAWABLE
drawable
4 GCONTEXT
gc
2 INT16
x
2 INT16
y
2n STRING16
string
p unused, p=pad(2n)

CreateColormap
1 78
1
0 None
1 All
2 4
4 COLORMAP
mid
4 WINDOW
window
4 VISUALID
visual

FreeColormap
1 79
1
2 2
4 COLORMAP
cmap

CopyColormapAndFree
1 80
1
2 3
4 COLORMAP
mid
4 COLORMAP
src-cmap

InstallColormap
1 81
1
2 2
4 COLORMAP
cmap

UninstallColormap
1 82
1
2 2
4 COLORMAP
cmap

ListInstalledColormaps
1 83
1
2 2
4 WINDOW
window

=>
1 1
Reply
1 unused
2   CARD16   sequence number
4   n      reply length
2   n      number of COLORMAPs in cmaps
22  unused
4n   LISTofCOLORMAP  cmaps

AllocColor
1   84      opcode
1    unused
2    4      request length
4   COLORMAP  cmap
2   CARD16   red
2   CARD16   green
2   CARD16   blue
2    unused

=>
1   1  Reply
1    unused
2   CARD16  sequence number
4   0      reply length
2   CARD16   red
2   CARD16   green
2   CARD16   blue
2    unused
4   CARD32   pixel
12  unused

AllocNamedColor
1   85      opcode
1    unused
2   3+(n+p)/4  request length
4   COLORMAP  cmap
2   n  length of name
2    unused
n   STRING8  name
p  unused, p=pad(n)

=>
1   1  Reply
1    unused
2   CARD16  sequence number
4   0      reply length
4   CARD32  pixel
2   CARD16  exact-red
2   CARD16  exact-green
2   CARD16  exact-blue
2   CARD16  visual-red
2   CARD16  visual-green
2   CARD16  visual-blue
8    unused

AllocColorCells
1   86      opcode
1   BOOL  contiguous
2    3      request length
4   COLORMAP  cmap
2   CARD16  colors
2   CARD16  planes

=>
1   1  Reply
1    unused
2   CARD16  sequence number
4   n+m      reply length
2   n  number of CARD32s in pixels
2   m  number of CARD32s in masks
X Protocol

Xll, Release 5

20
4n LISTofCARD32 unused
4m LISTofCARD32 pixels
4m LISTofCARD32 masks

AllocColorPlanes
1 87 opcode
1 BOOL contiguous
2 4 request length
4 COLORMAP cmap
2 CARD16 colors
2 CARD16 reds
2 CARD16 greens
2 CARD16 blues

=>
1 1 Reply
1 unused
2 CARD16 sequence number
4 n reply length
2 n number of CARD32s in pixels
2 unused
4 CARD32 red-mask
4 CARD32 green-mask
4 CARD32 blue-mask
8 unused
4n LISTofCARD32 pixels

FreeColors
1 88 opcode
1 unused
2 3+n request length
4 COLORMAP cmap
4 CARD32 plane-mask
4n LISTofCARD32 pixels

StoreColors
1 89 opcode
1 unused
2 2+3n request length
4 COLORMAP cmap
12n LISTofCOLORITEM items

COLORITEM
4 CARD32 pixel
2 CARD16 red
2 CARD16 green
2 CARD16 blue
1 do-red, do-green, do-blue
#x01 do-red (1 is True, 0 is False)
#x02 do-green (1 is True, 0 is False)
#x04 do-blue (1 is True, 0 is False)
#xF8 unused
1 unused

StoreNamedColor
1 90 opcode
1 do-red, do-green, do-blue
#x01 do-red (1 is True, 0 is False)
#x02 do-green (1 is True, 0 is False)
#x04 do-blue (1 is True, 0 is False)
#xF8 unused
2 4+(n+p)/4 request length
4 COLORMAP cmap
4 CARD32 pixel
2 n length of name
2 unused
n STRING8
p unused, p=pad(n)

QueryColors
1 91
1 unused
2 2+n request length
4 COLORMAP cmap
4n LISTofCARD32 pixels

=>
1 1 Reply
1 unused
2 CARD16 sequence number
4 2n reply length
2 n number of RGBs in colors
22 unused
8n LISTofRGB colors

RGB
2 CARD16 red
2 CARD16 green
2 CARD16 blue
2 unused

LookupColor
1 92
1 unused
2 3+(n+p)/4 request length
4 COLORMAP cmap
2 n length of name
2 unused
n STRING8 name
p unused, p=pad(n)

=>
1 1 Reply
1 unused
2 CARD16 sequence number
4 0 reply length
2 CARD16 exact-red
2 CARD16 exact-green
2 CARD16 exact-blue
2 CARD16 visual-red
2 CARD16 visual-green
2 CARD16 visual-blue
12 unused

CreateCursor
1 93
1 unused
2 8 request length
4 CURSOR cid
4 PIXMAP source
4 PIXMAP mask
0 None
2 CARD16 fore-red
2 CARD16 fore-green
2 CARD16 fore-blue
2 CARD16 back-red
2 CARD16 back-green
2 CARD16 back-blue
2 CARD16 x
2 CARD16 y

CreateGlyphCursor
1 94
1 unused

124
X Protocol

Xll, Release 5

1 unused
2 8 request length
4 CURSOR cid
4 FONT source-font
4 FONT mask-font
0 None
2 CARD16 source-char
2 CARD16 mask-char
2 CARD16 fore-red
2 CARD16 fore-green
2 CARD16 fore-blue
2 CARD16 back-red
2 CARD16 back-green
2 CARD16 back-blue

FreeCursor
1 95 opcode
1 unused
2 2 request length
4 CURSOR cursor

RecolorCursor
1 96 opcode
1 unused
2 5 request length
4 CURSOR cursor
2 CARD16 fore-red
2 CARD16 fore-green
2 CARD16 fore-blue
2 CARD16 back-red
2 CARD16 back-green
2 CARD16 back-blue

QueryBestSize
1 97 opcode
1 class
0 Cursor
1 Tile
2 Stipple
2 3 request length
4 DRAWABLE drawable
2 CARD16 width
2 CARD16 height
=>
1 1 Reply
1 unused
2 CARD16 sequence number
4 0 reply length
2 CARD16 width
2 CARD16 height
20 unused

QueryExtension
1 98 opcode
1 unused
2 2+(n+p)/4 request length
2 n length of name
2 unused
n STRING8 name
p unused, p=pad(n)
=>
1 1 Reply
1 unused
2 CARD16 sequence number
125
X Protocol

4 0
1 BOOL
1 CARD8
1 CARD8
1 CARD8 20

ListExtensions
1 99
1 unused
2 1 request length
=>
1 1
1 CARD8 2 CARD16 4 (n+p)/4 24
n LISTofSTR
p unused, p=pad(n)

ChangeKeyboardMapping
1 100
1 n keycode-count
2 2+nm request length
1 KEYCODE first-keycode
1 m keysyms-per-keycode
2 unused
4nm LISTofKEYSYM keysyms

GetKeyboardMapping
1 101
1 unused
2 2 request length
1 KEYCODE first-keycode
1 m count
2 unused
=>
1 1
1 n keysyms-per-keycode
2 CARD16 sequence number
4 nm reply length (m = count field from the request)
24 unused
4nm LISTofKEYSYM keysyms

ChangeKeyboardControl
1 102
1 unused
2 2+n request length
4 BITMASK
#x0001 key-click-percent
#x0002 bell-percent
#x0004 bell-pitch
#x0008 bell-duration
#x0100 led
#x0200 led-mode
#x0400 key
#x0800 auto-repeat-mode
4n LISTofVALUE value-list

VALUES
1 INT8 key-click-percent
1 INT8 bell-percent
2 INT16 bell-pitch
2 INT16 bell-duration
1 CARD8 led
X Protocol

1
0 Off
1 On

1 KEYCODE key
1 auto-repeat-mode
0 Off
1 On
2 Default

GetKeyboardControl
1 103 opcode
1 unused
2 1 request length
=>
1 1 Reply
1 0 Off
1 1 On
2 CARD16 sequence number
4 5 reply length
4 CARD32 led-mask
1 CARD8 key-click-percent
1 CARD8 bell-percent
2 CARD16 bell-pitch
2 CARD16 bell-duration
2 unused
32 LIST of CARD8 auto-repeats

Bell
1 104 opcode
1 INT8 percent
2 1 request length

ChangePointerControl
1 105 opcode
1 unused
2 3 request length
2 INT16 acceleration-numerator
2 INT16 acceleration-denominator
2 INT16 threshold
1 BOOL do-acceleration
1 BOOL do-threshold

GetPointerControl
1 106 opcode
1 unused
2 1 request length
=>
1 1 Reply
1 unused
2 CARD16 sequence number
4 0 reply length
2 CARD16 acceleration-numerator
2 CARD16 acceleration-denominator
2 CARD16 threshold
18 unused

SetScreenSaver
1 107 opcode
1 unused
2 3 request length
2 INT16 timeout
2 INT16 interval
1 prefer-blanking

127
0  No  allow-exposures
1  Yes
2  Default

1 0  No
1  Yes
2  Default

GetScreenSaver
1 108  opcode
1  unused
2  request length

=></>
1 1  Reply
1  unused
2  CARD16  sequence number
4 0  reply length
2  CARD16  timeout
2  CARD16  interval
1 0  No
1  Yes
1 0  No
1  Yes
18  unused

ChangeHosts
1 109  opcode
1  mode
0  Insert
1  Delete
2  2+(n+p)/4  request length
1  Internet
1  DECnet
2  Chaos
1  unused
2  n  length of address
n  LISTofCARD8  address
p  unused, p=pad(n)

ListHosts
1 110  opcode
1  unused
2  request length

=></>
1 1  Reply
1  mode
0  Disabled
1  Enabled
2  CARD16  sequence number
4  n/4  reply length
2  CARD16  number of HOSTs in hosts
22  unused
n  LISTofHOST  hosts (n always a multiple of 4)

SetAccessControl
1 111  opcode
1  mode
0  Disable
1  Enable
2  request length
X Protocol

Xll, Release 5

SetCloseDownMode
1 112 opcode
1 mode
0 Destroy
1 RetainPermanent
2 RetainTemporary
2 1 request length

KillClient
1 113 opcode
1 unused
2 2 request length
4 CARD32 resource
0 AllTemporary

RotateProperties
1 114 opcode
1 unused
2 3+n request length
4 WINDOW window
2 n number of properties
2 INT16 delta
4n LISTofATOM properties

ForceScreenSaver
1 115 opcode
1 unused
0 Reset
1 Activate
2 1 request length

SetPointerMapping
1 116 opcode
1 unused
n length of map
2 1+(n+p)/4 request length
n LISTofCARD8 map
p unused, p=pad(n)

=>
1 1 Reply
1 status
0 Success
1 Busy
2 CARD16 sequence number
4 0 reply length
24 unused

GetPointerMapping
1 117 opcode
1 unused
2 1 request length

=>
1 1 Reply
1 n length of map
2 CARD16 sequence number
4 (n+p)/4 reply length
24 unused
n LISTofCARD8 map
p unused, p=pad(n)

SetModifierMapping
1 118 opcode
1 n keycodes-per-modifier
2 1+2n request length
8n LISTofKEYCODE keycodes
X Protocol

X11, Release 5

GetModifierMapping
1 119
2 1

GetModifierMapping opacity
1 1
2 1

KeyPress
1 2
1 KEYCODE
2 CARD16
4 TIMESTAMP
4 WINDOW
4 WINDOW
0 None
2 INT16
2 INT16
2 INT16
2 INT16
2 SETofKEYBUTMASK
1 BOOL
1 unused

KeyRelease
1 3
1 KEYCODE
2 CARD16
4 TIMESTAMP
4 WINDOW
4 WINDOW
0 None
2 INT16
2 INT16
2 INT16
2 INT16
2 SETofKEYBUTMASK
1 BOOL
1 unused

ButtonPress
1 4
1 code
1   BUTTON     detail
2   CARD16    sequence number
4   TIMESTAMP  time
4   WINDOW    root
4   WINDOW    event
4   WINDOW    child
 0    None
2   INT16     root-x
2   INT16     root-y
2   INT16     event-x
2   INT16     event-y
2   SETofKEYBUTMASK state
1   BOOL      same-screen
1    unused

ButtonRelease
1   5         code
1   BUTTON    detail
2   CARD16    sequence number
4   TIMESTAMP  time
4   WINDOW    root
4   WINDOW    event
4   WINDOW    child
 0    None
2   INT16     root-x
2   INT16     root-y
2   INT16     event-x
2   INT16     event-y
2   SETofKEYBUTMASK state
1   BOOL      same-screen
1    unused

MotionNotify
1   6         code
1   BUTTON    detail
 0     Normal
     Hint
 1
2   CARD16    sequence number
4   TIMESTAMP  time
4   WINDOW    root
4   WINDOW    event
4   WINDOW    child
 0    None
2   INT16     root-x
2   INT16     root-y
2   INT16     event-x
2   INT16     event-y
2   SETofKEYBUTMASK state
1   BOOL      same-screen
1    unused

EnterNotify
1   7         code
1   BUTTON    detail
 0     Ancestor
     Virtual
     Inferior
     Nonlinear
     NonlinearVirtual
 2   CARD16    sequence number
4   TIMESTAMP  time
4   WINDOW    root
4   WINDOW    event
4   WINDOW    child
 0    None
2 INT16 root-x
2 INT16 root-y
2 INT16 event-x
2 INT16 event-y
2 SETofKEYBUTMASK state
1
  0 Normal
  1 Grab
  2 Ungrab
1
  same-screen, focus
  #x01 focus (1 is True, 0 is False)
  #x02 same-screen (1 is True, 0 is False)
  #xFc unused

LeaveNotify
1 8 code
1
  0 Ancestor
  1 Virtual
  2 Inferior
  3 Nonlinear
  4 NonlinearVirtual
2
  CARD16 sequence number
4
  TIMESTAMP time
4
  WINDOW root
4
  WINDOW event
4
  WINDOW child
  0 None
2 INT16 root-x
2 INT16 root-y
2 INT16 event-x
2 INT16 event-y
2 SETofKEYBUTMASK state
1
  mode
  0 Normal
  1 Grab
  2 Ungrab
1
  same-screen, focus
  #x01 focus (1 is True, 0 is False)
  #x02 same-screen (1 is True, 0 is False)
  #xFc unused

FocusIn
1 9 code
1
  0 Ancestor
  1 Virtual
  2 Inferior
  3 Nonlinear
  4 NonlinearVirtual
  5 Pointer
  6 PointerRoot
  7 None
2
  CARD16 sequence number
4
  WINDOW event
1
  mode
  0 Normal
  1 Grab
  2 Ungrab
  3 WhileGrabbed
23 unused

FocusOut
1 10 code
1
  detail
X Protocol

0  Ancestor
1  Virtual
2  Inferior
3  Nonlinear
4  Nonlinear Virtual
5  Pointer
6  Pointer Root
7  None

2  CARD16  sequence number
4  WINDOW  event
1  model

23  unused

KeymapNotify
1  11  code
31  LIST of CARD8  keys (byte for keycodes 0–7 is omitted)

Expose
1  12  code
1  unused
2  CARD16  sequence number
4  WINDOW  window
2  CARD16  x
2  CARD16  y
2  CARD16  width
2  CARD16  height
2  CARD16  count
14  unused

GraphicsExposure
1  13  code
1  unused
2  CARD16  sequence number
4  DRAWABLE  drawable
2  CARD16  x
2  CARD16  y
2  CARD16  width
2  CARD16  height
2  CARD16  minor-opcode
2  CARD16  count
1  CARD8  major-opcode
11  unused

NoExposure
1  14  code
1  unused
2  CARD16  sequence number
4  DRAWABLE  drawable
2  CARD16  minor-opcode
1  CARD8  major-opcode
21  unused

VisibilityNotify
1  15  code
1  unused
2  CARD16  sequence number
4  WINDOW  window
1  state
0  Unobscured
1  PartiallyObscured
2  FullyObscured
23  unused
<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
<th>Sequence Number</th>
<th>Parent</th>
<th>Window</th>
<th>Event</th>
<th>Width</th>
<th>Height</th>
<th>Border-Width</th>
<th>Override-Redirect</th>
<th>Unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateNotify</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DestroyNotify</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UnmapNotify</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MapNotify</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MapRequest</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReparentNotify</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConfigureNotify</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4 WINDOW
  0 None above-sibling
  1 INT16 x
  2 INT16 y
  3 CARD16 width
  4 CARD16 height
  5 CARD16 border-width
  6 BOOL override-redirect
  7 unused

ConfigureRequest
  1 23 code
  1 stack-mode
  0 Above
  1 Below
  2 TopIf
  3 BottomIf
  4 Opposite
  2 CARD16 sequence number
  4 WINDOW parent
  4 WINDOW window
  4 WINDOW sibling
  0 None
  2 INT16 x
  2 INT16 y
  2 CARD16 width
  2 CARD16 height
  2 CARD16 border-width
  2 BITMASK value-mask
    #x0001 x
    #x0002 y
    #x0004 width
    #x0008 height
    #x0010 border-width
    #x0020 sibling
    #x0040 stack-mode
  4 unused

GravityNotify
  1 24 code
  1 unused
  2 CARD16 sequence number
  4 WINDOW event
  4 WINDOW window
  2 INT16 x
  2 INT16 y
  16 unused

ResizeRequest
  1 25 code
  1 unused
  2 CARD16 sequence number
  4 WINDOW window
  4 WINDOW width
  2 CARD16 height
  20 unused

CirculatcNotify
  1 26 code
  1 unused
  2 CARD16 sequence number
  4 WINDOW event
  4 WINDOW window
  4 WINDOW unused
  1 place
CirculateRequest
0 Top unused
1 Bottom
15 unused

PropertyNotify
0 Top unused
1 Bottom
15 unused

SelectionClear
0 Top unused
1 Bottom
15 unused

SelectionRequest
0 Top unused
1 Bottom
15 unused

SelectionNotify
0 Top unused
1 Bottom
15 unused

ColormapNotify
0 Top unused
1 Bottom
15 unused

X Protocol
XI1, Release 5
1
2 CARD16 sequence number
4 WINDOW window
4 COLORMAP colormap
 0 None
1 BOOL new
1
0 Uninstalled
1 Installed
18 unused

ClientMessage
1 33 code
1 CARD8 format
2 CARD16 sequence number
4 WINDOW window
4 ATOM type
20 data

MappingNotify
1 34 code
1 unused
2 CARD16 sequence number
1
0 Modifier request
1 Keyboard
2 Pointer
1 KEYCODE first-keycode
1 CARD8 count
25 unused
Glossary

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

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

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

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

Background
An InputOutput window can have a background, which is defined as a pixmap. When regions of the window have their contents lost or invalidated, the server will automatically tile those regions with the background.

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

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

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

Bitmap
A bitmap is a pixmap of depth one.

Border
An InputOutput window can have a border of equal thickness on all four sides of the window. A pixmap defines the contents of the border, and the server automatically maintains the contents of the border. Exposure events are never generated for border regions.

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

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

Children
The children of a window are its first-level subwindows.
Client
An application program connects to the window system server by some interprocess com-
munication (IPC) path, such as a TCP connection or a shared memory buffer. This pro-
gram is referred to as a client of the window system server. More precisely, the client is
the IPC path itself; a program with multiple paths open to the server is viewed as multi-
ple clients by the protocol. Resource lifetimes are controlled by connection lifetimes, not
by program lifetimes.

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

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

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

Containment
A window “contains” the pointer if the window is viewable and the hotspot of the cur-
sor is within a visible region of the window or a visible region of one of its inferiors.
The border of the window is included as part of the window for containment. The
pointer is “in” a window if the window contains the pointer but no inferior contains the
pointer.

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

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

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

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

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

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

Drawable
Both windows and pixmaps can be used as sources and destinations in graphics opera-
tions. These windows and pixmaps are collectively known as drawables. However, an
InputOnly window cannot be used as a source or destination in a graphics operation.
Event

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

Event mask

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

Event synchronization

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

Event propagation

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

Event source

The window the pointer is in is the source of a device-related event.

Exposure event

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

Extension

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

Focus window

The focus window is another term for the input focus.

Font

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

GC, GContext

GC and gcontext are abbreviations for graphics context.

Glyph

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

Grab

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

Graphics context

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

Gravity

See bit gravity and window gravity.
Grayscale

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

Hotspot

A cursor has an associated hotspot that defines the point in the cursor corresponding to the coordinates reported for the pointer.

Identifier

An identifier is a unique value associated with a resource that clients use to name that resource. The identifier can be used over any connection.

Inferiors

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

Input focus

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

Input manager

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

InputOnly window

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

InputOutput window

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

Key grabbing

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

Keyboard grabbing

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

Keysym

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

Mapped

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

Modifier keys

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

Monochrome

Monochrome is a special case of StaticGray in which there are only two colormap entries.
Obscure

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

Occlude

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

Padding

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

Parent window

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

Passive grab

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

Pixel value

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

Pixmap

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

Plane

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

Plane mask

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

Pointer

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

Pointer grabbing

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

Pointing device

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

Property

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

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

PseudoColor

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

Redirecting control

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

Reply

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

Request

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

Resource

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

RGB values

Red, green, and blue (RGB) intensity values are used to define color. These values are always represented as 16-bit unsigned numbers, with 0 being the minimum intensity and 65535 being the maximum intensity. The server scales the values to match the display hardware.

Root

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

Root window

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

Save set

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

Scanline

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

Scanline order

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

Screen

A server can provide several independent screens, which typically have physically independent monitors. This would be the expected configuration when there is only a single keyboard and pointer shared among the screens.
Selection
A selection can be thought of as an indirect property with dynamic type; that is, rather than having the property stored in the server, it is maintained by some client (the “owner”). A selection is global in nature and is thought of as belonging to the user (although maintained by clients), rather than as being private to a particular window subhierarchy or a particular set of clients. When a client asks for the contents of a selection, it specifies a selection “target type”. This target type can be used to control the transmitted representation of the contents. For example, if the selection is “the last thing the user clicked on” and that is currently an image, then the target type might specify whether the contents of the image should be sent in XY format or Z format. The target type can also be used to control the class of contents transmitted; for example, asking for the “looks” (fonts, line spacing, indentation, and so on) of a paragraph selection rather than the text of the paragraph. The target type can also be used for other purposes. The protocol does not constrain the semantics.

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

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

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

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

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

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

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

String Equivalence
Two ISO Latin-1 STRING8 values are considered equal if they are the same length and if corresponding bytes are either equal or are equivalent as follows: decimal values 65 to 90 inclusive (characters “A” to “Z”) are pairwise equivalent to decimal values 97 to 122 inclusive (characters “a” to “z”), decimal values 192 to 214 inclusive (characters “A grave” to “O diaeresis”) are pairwise equivalent to decimal values 224 to 246 inclusive (characters “a grave” to “o diaeresis”), and decimal values 216 to 222 inclusive (characters “O oblique” to “THORN”) are pairwise equivalent to decimal values 246 to 254 inclusive (characters “o oblique” to “thorn”).

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

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

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

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

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

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

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

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

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

A
Above, 17, 19, 72
Access control list, 138
Access, 4, 15, 25, 26, 27, 28, 54, 62, 63, 98
Activate, 62
Active grab, 138
All, 50, 51, 54
Alloc, 4, 11, 20, 21, 32, 36, 41, 42, 50, 51, 52, 53, 55, 56, 57, 58, 99
AllocColor, 51, 52, 53, 54, 122
AllocColorCells, 51, 53, 54, 122
AllocColorPlanes, 51, 53, 54, 123
AllocNamedColor, 51, 52, 54, 122
AllowEvents, 24, 27, 28, 29, 110
AllTemporary, 63
AlreadyGrabbed, 24, 25, 26, 27
Alternative 1, 94
AlternativeValues, 1
Always, 8, 12, 14, 15, 69
Ancestor, 65, 66, 67
Ancestors, 138
And, 36, 37
AndInverted, 36, 37
AndReverse, 36, 37
AnyButton, 25, 26
AnyKey, 27, 28
AnyModifier, 25, 26, 27, 28
AnyPropertyType, 21, 22
Append, 21
AsyncBoth, 28, 29
Asynchronous, 24, 25, 26, 27, 64
AsyncKeyboard, 28, 29
AsyncPointer, 28, 29
Atom, 1, 4, 5, 21, 22, 23, 98, 138

B
Background, 138
Backing store, 138
Bell, 60, 127
Below, 17, 19, 72
Bevel, 36, 39
Bit:
  gravity, 138
  plane, 138
Bitmap, 48, 138
Border, 138
Bottom, 72
BottomIf, 17, 19, 72
Busy, 57, 60
Butt, 36, 38, 39, 41
Button1, 2
Button1Motion, 2
Button2, 2
Button2Motion, 2
Button3, 2
Button3Motion, 2
Button4, 2
Button4Motion, 2
Button5, 2
Button5Motion, 2
Button:
  grabbing, 138
ButtonMotion, 2, 65
ButtonDown, 2, 15, 25, 28, 29, 64, 65, 66, 130
ButtonRelease, 2, 28, 29, 64, 65, 66, 131
Button[1-5]Motion, 65
Byte order, 138

C
Center, 2, 18
ChangeActivePointerGrab, 26, 64, 109
ChangeGC, 4, 41, 42, 117
ChangeHosts, 4, 62, 128
ChangeKeyPressed, 4, 59, 126
ChangeKeyboardMapping, 58, 74, 126
ChangePointerControl, 61, 127
ChangeProperty, 21, 73, 106
ChangeSaveSet, 16, 104
ChangeWindowAttributes, 4, 11, 14, 51, 103
Chaos, 2
Children, 138
Chord, 36, 47
CirculateNotify, 20, 66, 69, 70, 72, 135
CirculateRequest, 20, 72, 136
CirculateWindow, 20, 72, 105
Clear, 36, 37
ClearArea, 43, 117
Client, 138
ClientMessage, 74, 137
ClipByChildren, 36, 40, 41, 43
Clipping region, 139
CloseFont, 32, 113
Colormap, 1, 4, 5, 11, 15, 51, 52, 53, 54, 55, 99, 139
ColormapChange, 2, 74
ColormapNotify, 15, 51, 52, 73, 136
Complex, 46, 47
ConfigureNotify, 18, 19, 66, 69, 70, 71, 134
ConfigureRequest, 18, 72, 135
ConfigureWindow, 14, 17, 71, 72, 105

146
X Protocol

Connection, 139
Containment, 139
Control, 2, 57, 58
ConvertSelection, 23, 73, 108
Convex, 46, 47
Coordinate system, 139
Copy, 36, 37, 41, 43, 50
CopyArea, 40, 43, 44, 70, 118
CopyColormapAndFree, 51, 121
CopyFromParent, 11, 12, 13, 14, 15, 93
CopyGC, 41, 117
CopyInverted, 36, 37
CopyPlane, 40, 43, 70, 118
CreateColormap, 50, 51, 54, 121
CreateCursor, 55, 56, 124
CreateGC, 36, 41, 42, 115
CreateGlyphCursor, 56, 124
CreateNotify, 14, 70, 134
CreatePixmap, 36, 115
CreateWindow, 11, 15, 51, 70, 102
CurrentTime, 23, 24, 25, 26, 27, 28, 30, 31, 73, 144
Cursor, 1, 4, 5, 11, 15, 24, 25, 26, 56, 98, 139

D

DECnet, 2
Default, 59, 61
Delete, 16, 62
Deleted, 73
DeleteProperty, 21, 73, 107
Depth, 139
Destroy, 63, 64
DestroyNotify, 16, 70, 134
DestroySubwindows, 16, 104
DestroyWindow, 16, 104
Device, 139
DirectColor, 8, 11, 51, 53, 139, 145
Disable, 63
Disabled, 62
Display, 139
DoubleDash, 36, 38, 39, 40
Drawable, 1, 4, 5, 20, 36, 43, 44, 45, 46, 47, 48, 49, 50, 56, 98, 139

E

East, 2, 18
Enable, 63
Enabled, 62
EnterNotify, 25, 65, 66, 67, 69, 70, 131
EnterWindow, 2, 65
Equiv, 36, 37

Error Codes:
 Access, 4
 Alloc, 4
 Atom, 4
 Colormap, 4
 Cursor, 4
 Drawable, 4
 Font, 4
 GContext, 4
 IDChoice, 5
 Implementation, 5
 Length, 5
 Match, 5
 Name, 5
 Pixmaps, 5
 Request, 5
 Value, 5
 Window, 5
 EvenOdd, 36, 40, 41
 Event, 139
 Exposure, 140
 mask, 140
 propagation, 140
 source, 140
 synchronization, 140
 EventName, 2
 Expose, 66, 67, 69, 70, 133
 Exposure, 2, 69
 Extension, 140

F

Failed, 57
False, 2, 12, 17, 18, 20, 24, 27, 30, 64, 66, 71, 74
FillPoly, 39, 40, 46, 47, 119
Focus window, 140
FocusChange, 2, 67
FocusIn, 27, 31, 67, 69, 132
FocusOut, 27, 31, 67, 69, 70, 132
Font, 1, 4, 5, 32, 33, 34, 36, 41, 49, 56, 98, 140
ForceScreenSaver, 61, 62, 129
Forget, 2, 12, 19
Free, 63
FreeColormap, 14, 51, 121
FreeColors, 4, 51, 54, 123
FreeCursor, 56, 125
FreeGC, 43, 117
FreePixmap, 36, 115
Frozen, 24, 25, 26, 27
FullyObscured, 70

G

GC, 140
GContext, 1, 4, 5, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 99, 140
GetAtomName, 21, 106
GetFontPath, 36, 115
GetGeometry, 20, 105
GetImage, 48, 120
GetInputFocus, 31, 112
GetKeyboardControl, 60, 127
GetKeyboardMapping, 58, 126
GetModifierMapping, 57, 130
GetMotionEvents, 10, 30, 65, 111
GetPointerControl, 61, 127
GetPointerMapping, 60, 61, 129
GetProperty, 21, 73, 107
GetScreenSaver, 61, 128
GetSelectionOwner, 23, 107
GetWindowAttributes, 11, 15, 103
Glyph, 140
Grab, 65, 66, 67, 68, 140
GrabButton, 25, 26, 29, 64, 109
GrabKey, 27, 28, 29, 110
GrabKeyboard, 26, 27, 28, 29, 109
GrabPointer, 24, 25, 26, 29, 108
GrabServer, 29, 110
Graphics context, 140
GraphicsExposure, 40, 43, 69, 70, 133
Gravity, 140
GravityNotify, 19, 66, 69, 70, 135
Grayscale, 8, 11, 51, 53, 55, 140, 144

H

Hint, 65
Hotspot, 141

I

IDChoice, 1, 4, 11, 32, 36, 50, 51, 55, 99
Identifier, 141
ImageText16, 50, 121
ImageText8, 50, 121
Implementation, 4, 99
IncludeInferiors, 36, 40
Inferior, 65, 66, 67
Inferiors, 141
Input focus, 141
Input manager, 141
InputFocus, 23, 24, 94
InputOnly, 4, 10, 11, 12, 13, 15, 17, 18, 20, 36, 43, 56, 69, 70, 93, 139, 141
InputOutput, 10, 11, 12, 15, 40, 93, 138, 141
Insert, 16, 62
InstallColormap, 10, 14, 15, 51, 52, 121
Installed, 74
InternAtom, 6, 20, 106
Internet, 2
InvalidTime, 24, 25, 26, 27
Invert, 36, 37

K

Key:
  grabbing, 141
Keyboard, 74
  grabbing, 141
KeymapNotify, 1, 69, 133
KeymapState, 2, 69
KeyPress, 2, 6, 27, 29, 59, 64, 65, 68, 130
KeyRelease, 2, 27, 29, 59, 64, 65, 69, 130
Keysym, 141
KillClient, 63, 129

L

LeastSignificant, 8
LeaveNotify, 25, 65, 66, 67, 69, 70, 132
LeaveWindow, 2, 65
LeftToRight, 32, 33, 34
Length, 4, 10, 57, 58, 99
ListExtensions, 57, 126
ListFonts, 35
ListFonts, 35, 114
ListFontsWithInfo, 35, 114
ListHosts, 62, 128
ListInstalledColormaps, 52, 121
ListProperties, 22, 107
Lock, 2, 57, 58
LookupColor, 55, 124
LowerHighest, 20
LSBFirst, 8

M

MapNotify, 17, 66, 69, 70, 71, 134
Mapped window, 141
MappingNotify, 57, 58, 60, 74, 137
MapRequest, 17, 71, 134
MapSubwindows, 17, 104
MapWindow, 16, 17, 63, 71, 104
Match, 4, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 31, 36, 39, 40, 41, 42, 43, 54, 45, 46, 47, 48, 49, 50, 51, 55, 56, 59, 98
Miter, 36, 39, 41
Mod1, 2, 57, 58
Mod2, 2, 57, 58
X Protocol

Mod3, 2, 57, 58
Mod4, 2, 57, 58
Mod5, 2, 57, 58
Modifier keys, 141
Modifier, 74
Modifiers, 74
Monochrome, 141
MostSignificant, 8
MotionNotify, 10, 64, 65, 131
MSBFirst, 8

N

Name1, 93
Name, 4, 32, 53, 54, 55, 99
Name1, 93
NameOfThing, 93
Nand, 36, 37
Never, 8
NewValue, 73
No, 61, 62
NoExposure, 43, 70, 133
Nonconvex, 46, 47
None, 5, 12, 13, 14, 15, 20, 21, 22, 23, 24, 25, 26, 30, 31, 32, 36, 40, 41, 42, 43, 48, 50, 51, 55, 56, 64, 65, 66, 67, 68, 71, 72, 73
Nonlinear, 65, 66, 67, 68
NonlinearVirtual, 65, 66, 67, 68
NoOp, 36, 37
NoOperation, 63, 130
Nor, 36, 37
Normal, 65, 66, 67, 68, 69
North, 2, 18
NorthEast, 2, 18
NorthWest, 2, 12, 18, 19
NoSymbol, 58, 59
NotLast, 36, 38, 39
NotUseful, 12, 14, 15
NotViewable, 24, 25, 26, 27

O

Obscure, 141
Oclude, 142
Off, 59, 60
On, 59, 60
OnOffDash, 36, 38, 39
OpaqueStippled, 36, 39, 40, 44
OpenFont, 32, 112
Opposite, 17, 19, 72
Or, 36, 37
Origin, 44, 46
OrInverted, 36, 37
OrReverse, 36, 37
OwnerGrabButton, 2, 64

P

Padding, 142
Parent, 31, 32
ParentRelative, 12, 13, 15, 17
PartiallyObscured, 70
Passive grab, 142
PicSlice, 36, 41, 47
Pixel value, 142
Pixmap, 1, 4, 5, 11, 15, 36, 41, 55, 98, 142
Plane, 142
mask, 142
Pointer, 67, 68, 74, 142
grabbing, 142
PointerMotion, 2, 65
PointerMotionHint, 2, 65
PointerRoot, 5, 31, 32, 64, 67, 68
PointerWindow, 23, 94
Pointing device, 142
PolyArc, 39, 45, 47, 119
PolyFillArc, 39, 40, 47, 119
PolyFillRectangle, 39, 47, 119
PolyLine, 39, 44, 45, 118
PolyPoint, 44, 118
PolyRectangle, 39, 45, 119
PolySegment, 39, 45, 118
PolyText16, 4, 39, 49, 120
PolyText8, 4, 39, 49, 120
Prepend, 21
Previous, 44, 46
Projecting, 36, 38, 39
Property list, 142
Property, 142
PropertyChange, 2, 73
PropertyNotify, 21, 22, 73, 136
PseudoColor, 8, 11, 51, 53, 140, 143, 144
PutImage, 48, 119

Q

QueryBestSize, 56, 125
QueryColors, 54, 124
QueryExtension, 57, 125
QueryFont, 32, 34, 35, 113
QueryKeymap, 32, 69, 112
QueryPointer, 30, 65, 111
QueryTextExtents, 34, 50, 113
QueryTree, 20, 105

R

RaiseLowest, 20
RecolorCursor, 56, 125
Redirecting control, 143
ReparentNotify, 16, 71, 134
ReparentWindow, 16, 104
Replace, 21
ReplayKeyboard, 28, 29
ReplayPointer, 28, 29
Reply, 143
Request, 4, 97, 143
RequestName, 1
Reset, 61, 62
ResizeRedirect, 2, 15, 18, 72
ResizeRequest, 18, 72, 135
Resource, 143
RetainPermanent, 63, 64
RetainTemporary, 63, 64
RGB values, 143
RightToLeft, 32, 33, 34
Root, 143
RotateProperties, 22, 73, 129
Round, 36, 38, 39

S
Save set, 143
Scanline order, 143
Scanline, 143
Screen, 143
Selection, 143
SelectionClear, 23, 73, 136
SelectionNotify, 23, 73, 136
SelectionRequest, 23, 73, 136
SendEvent, 1, 23, 73, 74, 108
Server, 144
    grabbing, 144
Set, 36, 37
SetAccessControl, 63, 128
SetClipRectangles, 40, 41, 42, 117
SetCloseDownMode, 63, 129
SetDashes, 40, 41, 42, 117
SetFontPath, 35, 115
SetInputFocus, 31, 67, 112
SetModifierMapping, 57, 74, 129
SetPointerMapping, 60, 74, 129
SetScreenSaver, 61, 127
SetSelectionOwner, 22, 63, 73, 107
Shift, 2, 57, 58
Sibling, 144
Solid, 36, 38, 39, 40, 41, 50
South, 2, 18
Southeast, 2, 18
SouthWest, 2, 18
Stacking order, 144
Static, 2, 19
StaticColor, 8, 11, 51, 144
StaticGray, 8, 11, 51, 55, 141, 144
Stipple, 56, 144
Stippled, 36, 39, 40
StoreColors, 4, 53, 54, 123
StoreNamedColor, 53, 54, 123
String Equivalence, 144
StructureNotify, 2, 70, 71, 72
SubstructureNotify, 2, 70, 71, 72
SubstructureRedirect, 2, 14, 15, 17, 18, 20, 71, 72
Success, 24, 26, 57, 60
SyncBoth, 28, 29
Synchronous, 24, 25, 26, 27
SyncKeyboard, 28, 29
SyncPointer, 28, 29

T
Tile, 56, 144
Tiled, 36, 39, 40
Timestamp, 144
Top, 72
TopIf, 17, 19, 72
TranslateCoordinates, 30, 111
True, 2, 10, 14, 16, 22, 24, 27, 30, 33, 41, 43, 53, 64, 66, 71, 74
TrueColor, 8, 11, 51, 145
Type, 145
Types:
    ARC, 3
    ATOM, 3
    BITGRAVITY, 3
    BITMASK, 2
    BOOL, 3
    BUTMASK, 3
    BUTTON, 3
    BYTE, 3
    CARD16, 3
    CARD32, 3
    CARD8, 3
    CHAR2B, 3
    COLORMAP, 2
    CURSOR, 2
    DEVICEEVENT, 3
    DRAWABLE, 2
    EVENT, 3
    FONT, 2
    FONTABLE, 3
    GCONTEXT, 2
    HOST, 3
    INT16, 3
    INT32, 3
    INT8, 3
    KEYBUTMASK, 3
    KEYCODE, 3
    KEYPESK, 3
KEYSYM, 3
LISTToFOO, 2
LISTToVALUE, 2
OR, 2
PIXMAP, 2
POINT, 3
POINTEREVENT, 3
RECTANGLE, 3
STRING16, 3
STRING8, 3
TIMESTAMP, 3
VALUE, 3
VISUALID, 3
WINDOW, 2
WINGRAVITY, 3

U

Ungrab, 65, 66, 67, 69
UngrabButton, 26, 109
UngrabKey, 28, 110
UngrabKeyboard, 27, 63, 110
UngrabPointer, 25, 63, 64, 109
UngrabServer, 30, 63, 111
UninstallColormap, 51, 52, 121
Uninstalled, 74
Unmap, 2, 19, 71
UnmapNotify, 17, 19, 66, 67, 69, 70, 71, 134
Unmapped, 15
UnmapSubwindows, 17, 104
UnmapWindow, 16, 17, 104
Unobscured, 70
Unsorted, 42
Unviewable, 15

V

Value, 1, 2, 4, 11, 12, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28, 31, 35, 36, 40, 41, 42, 43, 44, 46, 48, 50, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 97
Viewable, 15, 145
Virtual, 65, 66, 67
Visibility, 69
VisibilityChange, 2, 70
VisibilityNotify, 12, 66, 67, 69, 70, 133
Visible, 145

W

WarpPointer, 31, 112
West, 2, 18
WhenMapped, 8, 12, 14, 15, 69
WhileGrabbed, 67
Winding, 36, 40
Xlib – C Language X Interface
MIT X Consortium Standard
X Version 11, Release 5
First Revision - August, 1991

James Gettys
Cambridge Research Laboratory
Digital Equipment Corporation

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

with contributions from

Chuck Adams, Tektronix, Inc.
Vania Joloboff, Open Software Foundation
Bill McMahon, Hewlett-Packard Company
Ron Newman, Massachusetts Institute of Technology
Al Tabayoyon, Tektronix, Inc.
Glenn Widener, Tektronix, Inc.
Acknowledgments

The design and implementation of the first 10 versions of X were primarily the work of three individuals: Robert Scheifler of the MIT Laboratory for Computer Science and Jim Gettys of Digital Equipment Corporation and Ron Newman of MIT, both at MIT Project Athena. X version 11, however, is the result of the efforts of dozens of individuals at almost as many locations and organizations. At the risk of offending some of the players by exclusion, we would like to acknowledge some of the people who deserve special credit and recognition for their work on Xlib. Our apologies to anyone inadvertently overlooked.

Release 1

Our thanks does to Ron Newman (MIT Project Athena), who contributed substantially to the design and implementation of the Version 11 Xlib interface.

Our thanks also goes to Ralph Swick (Project Athena and Digital) who kept it all together for us during the early releases. He handled literally thousands of requests from people everywhere and saved the sanity of at least one of us. His calm good cheer was a foundation on which we could build.

Our thanks also goes to Todd Brunhoff (Tektronix) who was “loaned” to Project Athena at exactly the right moment to provide very capable and much-needed assistance during the alpha and beta releases. He was responsible for the successful integration of sources from multiple sites; we would not have had a release without him.

Our thanks also goes to Al Mento and Al Wojtas of Digital’s ULTRIX Documentation Group. With good humor and cheer, they took a rough draft and made it an infinitely better and more useful document. The work they have done will help many everywhere. We also would like to thank Hal Murray (Digital SRC) and Peter George (Digital VMS) who contributed much by proofreading the early drafts of this document.

Our thanks also goes to Jeff Dike (Digital UEG), Tom Benson, Jackie Granfield, and Vince Orgovan (Digital VMS) who helped with the library utilities implementation; to Hania Gajewska (Digital UEG-WSL) who, along with Ellis Cohen (CMU and Siemens), was instrumental in the semantic design of the window manager properties; and to Dave Rosenthal (Sun Microsystems) who also contributed to the protocol and provided the sample generic color frame buffer device-dependent code.

The alpha and beta test participants deserve special recognition and thanks as well. It is significant that the bug reports (and many fixes) during alpha and beta test came almost exclusively from just a few of the alpha testers, mostly hardware vendors working on product implementations of X. The continued public contribution of vendors and universities is certainly to the benefit of the entire X community.

Our special thanks must go to Sam Fuller, Vice-President of Corporate Research at Digital, who has remained committed to the widest public availability of X and who made it possible to greatly supplement MIT’s resources with the Digital staff in order to make version 11 a reality. Many of the people mentioned here are part of the Western Software Laboratory (Digital UEG-WSL) of the ULTRIX Engineering group and work for Smokey Wallace, who has been vital to the project’s success. Others not mentioned here worked on the toolkit and are acknowledged in the X Toolkit documentation.

Of course, we must particularly thank Paul Asente, formerly of Stanford University and now of Digital UEG-WSL, who wrote W, the predecessor to X, and Brian Reid, formerly of Stanford University and now of Digital WRL, who had much to do with W’s design.

Finally, our thanks goes to MIT, Digital Equipment Corporation, and IBM for providing the environment where it could happen.
Release 4

Our thanks go to Jim Fulton (MIT X Consortium) for designing and specifying the new Xlib functions for Inter-Client Communication Conventions (ICCCM) support.

We also thank Al Mento of Digital for his continued effort in maintaining this document and Jim Fulton and Donna Converse (MIT X Consortium) for their much-appreciated efforts in reviewing the changes.

Release 5

The principal authors of the Input Method facilities are Vania Joloboff (Open Software Foundation) and Bill McMahon (Hewlett-Packard). The principal author of the rest of the internationalization facilities is Glenn Widener (Tektronix). Our thanks to them for keeping their sense of humor through a long and sometimes difficult design process. Although the words and much of the design are due to them, many others have contributed substantially to the design and implementation. Tom McFarland (HP) and Frank Rojas (IBM) deserve particular recognition for their contributions. Other contributors were: Tim Anderson (Motorola), Alka Badshah (OSF), Gabe Beged-Dov (HP), Chih-Chung Ko (III), Vera Cheng (III), Michael Collins (Digital), Walt Daniels (IBM), Noritoshi Demizu (OMRON), Keisuke Fukui (Fujitsu), Hitoshi Fukumoto (Nihon Sun), Tim Greenwood (Digital), John Harvey (IBM), Fred Hornman (AT&T), Norikazu Kaima (Fujitsu), Yuji Kamata (IBM), Yutaka Kataoka (Waseda University), Raneen Khubchandani (Sun), Akira Kon (NEC), Hiroshi Kuribayashi (OMRON), Teruhiko Kurosaka (Sun), Seiji Kuwari (OMRON), Sandra Martin (OSF), Masato Morisaki (NTT), Nelson Ng (Sun), Takashi Nishimura (NTT America), Makato Nishino (IBM), Akira Ohsome (Nihon Sun), Chris Peterson (MIT), Sam Shteingart (AT&T), Manish Sheth (AT&T), Muneiyoshi Suzuki (NTT), Cori Mehring (Digital), Shoji Sugiyama (IBM), and Eiji Tosa (IBM).

We are deeply indebted to Tatsuya Kato (NTT), Hiroshi Kuribayashi (OMRON), Seiji Kuwari (OMRON), Muneiyoshi Suzuki (NTT), and Li Yuhong (OMRON) for producing the first complete sample implementation of the internationalization facilities. We are also very much indebted to Masato Morisaki (NTT) for coordinating the integration, testing, and release of this implementation. We also thank Michael Collins for his design of the "pluggable layer" inside Xlib.

The principal authors (design and implementation) of the Xcms color management facilities are Al Tabayoyon (Tektronix) and Chuck Adams (Tektronix). Joann Taylor (Tektronix), Bob Toole (Tektronix), and Keith Packard (MIT X Consortium) also contributed significantly to the design. Others who contributed are: Harold Boll (Kodak), Ken Bronstein (HP), Nancy Cam (SGI), Donna Converse (MIT X Consortium), Elias Israel (ISC), Deron Johnson (Sun), Jim King (Adobe), Ricardo Motta (HP), Keith Packard (MIT), Chuck Peck (IBM), Wil Plouffe (IBM), Dave Stemlicht (MIT X Consortium), Kumar Talluri (AT&T), and Richard Verberg (IBM).

We also once again thank Al Mento of Digital for his work in formatting and reformatting text for this manual, and for producing man pages. Thanks also to Clive Feather (IXI) for proofreading and finding a number of small errors.

Jim Gettys
Cambridge Research Laboratory
Digital Equipment Corporation

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

Table of Contents .................................................................................................................. ii
Acknowledgments .................................................................................................................. iii
Chapter 1: Introduction to Xlib ............................................................................................... 1
  1.1. Overview of the X Window System .............................................................................. 1
  1.2. Errors ......................................................................................................................... 2
  1.3. Standard Header Files ............................................................................................... 3
  1.4. Generic Values and Types ......................................................................................... 4
  1.5. Naming and Argument Conventions within Xlib ..................................................... 4
  1.6. Programming Considerations .................................................................................. 5
  1.7. Character Sets and Encodings .................................................................................. 5
  1.8. Formatting Conventions ........................................................................................... 6
Chapter 2: Display Functions .................................................................................................... 7
  2.1. Opening the Display .................................................................................................. 7
  2.2. Obtaining Information about the Display, Image Formats, or Screens .................... 8
      2.2.1. Display Macros .................................................................................................. 8
      2.2.2. Image Format Functions and Macros ................................................................ 14
      2.2.3. Screen Information Macros ........................................................................... 16
  2.3. Generating a NoOperation Protocol Request ......................................................... 19
  2.4. Freeing Client-Created Data .................................................................................... 19
  2.5. Closing the Display .................................................................................................. 19
  2.6. X Server Connection Close Operations .................................................................... 20
Chapter 3: Window Functions .................................................................................................. 22
  3.1. Visual Types .............................................................................................................. 22
  3.2. Window Attributes .................................................................................................... 23
      3.2.1. Background Attribute ..................................................................................... 25
      3.2.2. Border Attribute .............................................................................................. 26
      3.2.3. Gravity Attributes ......................................................................................... 27
      3.2.4. Backing Store Attribute .................................................................................. 28
      3.2.5. Save Under Flag ......................................................................................... 28
      3.2.6. Backing Planes and Backing Pixel Attributes .................................................. 28
      3.2.7. Event Mask and Do Not Propagate Mask Attributes ...................................... 28
      3.2.8. Override Redirect Flag .................................................................................... 29
      3.2.9. Colormap Attribute .......................................................................................... 29
      3.2.10. Cursor Attribute ............................................................................................ 29
  3.3. Creating Windows ...................................................................................................... 29
  3.4. Destroying Windows .................................................................................................. 32
6.12. Color Management Extensions .......................................................... 95
  6.12.1. Color Spaces ............................................................................ 95
  6.12.2. Adding Device-Independent Color Spaces ................................. 96
  6.12.3. Querying Color Space Format and Prefix ................................. 96
  6.12.4. Creating Additional Color Spaces ............................................ 96
  6.12.5. Parse String Callback ............................................................... 97
  6.12.7. Function Sets .......................................................................... 99

Chapter 7: Graphics Context Functions .................................................. 101
  7.1. Manipulating Graphics Context/State ........................................... 101
  7.2. Using GC Convenience Routines .................................................. 109
    7.2.1. Setting the Foreground, Background, Function, or Plane Mask .. 109
    7.2.2. Setting the Line Attributes and Dashes .................................... 111
    7.2.3. Setting the Fill Style and Fill Rule ......................................... 112
    7.2.4. Setting the Fill Tile and Stipple .............................................. 112
    7.2.5. Setting the Current Font ....................................................... 114
    7.2.6. Setting the Clip Region ......................................................... 115
    7.2.7. Setting the Arc Mode, Subwindow Mode, and Graphics Exposure .. 116

Chapter 8: Graphics Functions ............................................................. 118
  8.1. Clearing Areas ............................................................................... 118
  8.2. Copying Areas .............................................................................. 119
  8.3. Drawing Points, Lines, Rectangles, and Arcs ................................. 120
    8.3.1. Drawing Single and Multiple Points ....................................... 121
    8.3.2. Drawing Single and Multiple Lines ......................................... 122
    8.3.3. Drawing Single and Multiple Rectangles ................................. 123
    8.3.4. Drawing Single and Multiple Arcs ......................................... 124
  8.4. Filling Areas .................................................................................. 126
    8.4.1. Filling Single and Multiple Rectangles .................................... 126
    8.4.2. Filling a Single Polygon ......................................................... 127
    8.4.3. Filling Single and Multiple Arcs .............................................. 128
  8.5. Font Metrics .................................................................................. 129
    8.5.1. Loading and Freeing Fonts ..................................................... 132
    8.5.2. Obtaining and Freeing Font Names and Information ................. 134
    8.5.3. Computing Character String Sizes ........................................ 135
    8.5.4. Computing Logical Extents .................................................... 136
    8.5.5. Querying Character String Sizes ........................................... 137
  8.6. Drawing Text ................................................................................ 138
    8.6.1. Drawing Complex Text ............................................................ 139
    8.6.2. Drawing Text Characters ........................................................ 140
    8.6.3. Drawing Image Text Characters .............................................. 141
8.7. Transferring Images between Client and Server ................................................................. 142
Chapter 9: Window and Session Manager Functions ................................................................. 147
9.1. Changing the Parent of a Window ......................................................................................... 147
9.2. Controlling the Lifetime of a Window .................................................................................. 148
9.3. Managing Installed Colormaps ............................................................................................ 149
9.4. Setting and Retrieving the Font Search Path ..................................................................... 150
9.5. Server Grabbing .................................................................................................................. 151
9.6. Killing Clients ..................................................................................................................... 151
9.7. Screen Saver Control ........................................................................................................... 152
9.8. Controlling Host Access ...................................................................................................... 153
9.8.1. Adding, Getting, or Removing Hosts .............................................................................. 154
9.8.2. Changing, Enabling, or Disabling Access Control ......................................................... 156
Chapter 10: Events ....................................................................................................................... 157
10.1. Event Types ....................................................................................................................... 157
10.2. Event Structures ............................................................................................................... 158
10.3. Event Masks ....................................................................................................................... 159
10.4. Event Processing Overview .............................................................................................. 160
10.5. Keyboard and Pointer Events ............................................................................................ 162
10.5.1. Pointer Button Events .................................................................................................. 162
10.5.2. Keyboard and Pointer Events ....................................................................................... 163
10.6. Window Entry/Exit Events ................................................................................................. 165
10.6.1. Normal Entry/Exit Events ............................................................................................ 166
10.6.2. Grab and Ungrab Entry/Exit Events .............................................................................. 167
10.7. Input Focus Events ............................................................................................................. 168
10.7.1. Normal Focus Events and Focus Events While Grabbed .............................................. 169
10.7.2. Focus Events Generated by Grabs ................................................................................ 171
10.8. Key Map State Notification Events .................................................................................... 171
10.9. Exposure Events ............................................................................................................... 172
10.9.1. Expose Events .............................................................................................................. 172
10.9.2. GraphicsExpose and NoExpose Events ....................................................................... 173
10.10. Window State Change Events ........................................................................................... 174
10.10.1. CirculateNotify Events .............................................................................................. 174
10.10.2. ConfigureNotify Events ............................................................................................ 175
10.10.3. CreateNotify Events .................................................................................................. 175
10.10.4. DestroyNotify Events ................................................................................................ 176
10.10.5. GravityNotify Events ................................................................................................. 176
10.10.6. MapNotify Events ...................................................................................................... 177
10.10.7. MappingNotify Events ............................................................................................... 178
10.10.8. ReparentNotify Events ............................................................................................... 178
10.10.9. UnmapNotify Events .................................................................................................. 179
10.10.10. VisibilityNotify Events ............................................................................................. 179
10.11. Structure Control Events ................................................................................................ 180

vi
10.11.1. CirculateRequest Events .............................................. 180
10.11.2. ConfigureRequest Events ........................................... 181
10.11.3. MapRequest Events .................................................. 181
10.11.4. ResizeRequest Events ............................................... 182
10.12. Colormap State Change Events ....................................... 182
10.13. Client Communication Events ....................................... 183
10.13.1. ClientMessage Events .............................................. 183
10.13.2. PropertyNotify Events ............................................. 184
10.13.3. SelectionClear Events ............................................. 184
10.13.4. SelectionRequest Events .......................................... 185
10.13.5. SelectionNotify Events ............................................ 185
Chapter 11: Event Handling Functions ................................... 187
11.1. Selecting Events ....................................................... 187
11.2. Handling the Output Buffer ......................................... 188
11.3. Event Queue Management .............................................. 188
11.4. Manipulating the Event Queue ....................................... 189
11.4.1. Returning the Next Event ......................................... 189
11.4.2. Selecting Events Using a Predicate Procedure ............ 190
11.4.3. Selecting Events Using a Window or Event Mask .......... 191
11.5. Putting an Event Back into the Queue .............................. 193
11.6. Sending Events to Other Applications ............................ 194
11.7. Getting Pointer Motion History .................................... 195
11.8. Handling Protocol Errors ............................................ 195
11.8.1. Enabling or Disabling Synchronization ......................... 196
11.8.2. Using the Default Error Handlers ............................... 196
Chapter 12: Input Device Functions ...................................... 201
12.1. Pointer Grabbing ....................................................... 201
12.2. Keyboard Grabbing ...................................................... 205
12.3. Resuming Event Processing .......................................... 208
12.4. Moving the Pointer ...................................................... 210
12.5. Controlling Input Focus ............................................... 211
12.6. Keyboard and Pointer Settings ..................................... 212
12.7. Keyboard Encoding ...................................................... 216
Chapter 13: Locales and Internationalized Text Functions ............ 222
13.1. X Locale Management .................................................. 222
13.2. Locale and Modifier Dependencies .................................. 224
13.3. Creating and Freeing a Font Set .................................... 225
13.4. Obtaining Font Set Metrics .......................................... 228
13.5. Drawing Text Using Font Sets ....................................... 233
13.6. Input Method Overview ............................................... 235
13.6.1. Input Method Architecture ........................................ 237
13.6.2. Input Contexts ....................................................... 238
Chapter 1

Introduction to Xlib

The X Window System is a network-transparent window system that was designed at MIT. X display servers run on computers with either monochrome or color bitmap display hardware. The server distributes user input to and accepts output requests from various client programs located either on the same machine or elsewhere in the network. Xlib is a C subroutine library that application programs (clients) use to interface with the window system by means of a stream connection. Although a client usually runs on the same machine as the X server it is talking to, this need not be the case.

Xlib – C Language X Interface is a reference guide to the low-level C language interface to the X Window System protocol. It is neither a tutorial nor a user's guide to programming the X Window System. Rather, it provides a detailed description of each function in the library as well as a discussion of the related background information. Xlib – C Language X Interface assumes a basic understanding of a graphics window system and of the C programming language. Other higher-level abstractions (for example, those provided by the toolkits for X) are built on top of the Xlib library. For further information about these higher-level libraries, see the appropriate toolkit documentation. The X Window System Protocol provides the definitive word on the behavior of X. Although additional information appears here, the protocol document is the ruling document.

To provide an introduction to X programming, this chapter discusses:

- Overview of the X Window System
- Errors
- Standard header files
- Naming and argument conventions
- Programming considerations
- Formatting conventions

1.1. Overview of the X Window System

Some of the terms used in this book are unique to X, and other terms that are common to other window systems have different meanings in X. You may find it helpful to refer to the glossary, which is located at the end of the book.

The X Window System supports one or more screens containing overlapping windows or subwindows. A screen is a physical monitor and hardware, which can be either color, grayscale, or monochrome. There can be multiple screens for each display or workstation. A single X server can provide display services for any number of screens. A set of screens for a single user with one keyboard and one pointer (usually a mouse) is called a display.

All the windows in an X server are arranged in strict hierarchies. At the top of each hierarchy is a root window, which covers each of the display screens. Each root window is partially or completely covered by child windows. All windows, except for root windows, have parents. There is usually at least one window for each application program. Child windows may in turn have their own children. In this way, an application program can create an arbitrarily deep tree on each screen. X provides graphics, text, and raster operations for windows.

A child window can be larger than its parent. That is, part or all of the child window can extend beyond the boundaries of the parent, but all output to a window is clipped by its parent. If several children of a window have overlapping locations, one of the children is considered to be on top of or raised over the others thus obscuring them. Output to areas covered by other
windows is suppressed by the window system unless the window has backing store. If a window is obscured by a second window, the second window obscures only those ancestors of the second window, which are also ancestors of the first window.

A window has a border zero or more pixels in width, which can be any pattern ( pixmap ) or solid color you like. A window usually but not always has a background pattern, which will be repainted by the window system when uncovered. Child windows obscure their parents, and graphic operations in the parent window usually are clipped by the children.

Each window and pixmap has its own coordinate system. The coordinate system has the X axis horizontal and the Y axis vertical, with the origin \([0, 0]\) at the upper left. Coordinates are integral, in terms of pixels, and coincide with pixel centers. For a window, the origin is inside the border at the inside upper left.

X does not guarantee to preserve the contents of windows. When part or all of a window is hidden and then brought back onto the screen, its contents may be lost. The server then sends the client program an Expose event to notify it that part or all of the window needs to be repainted. Programs must be prepared to regenerate the contents of windows on demand.

X also provides off-screen storage of graphics objects, called pixmaps. Single plane (depth 1) pixmaps are sometimes referred to as bitmaps. Pixmaps can be used in most graphics functions interchangeably with windows and are used in various graphics operations to define patterns or tiles. Windows and pixmaps together are referred to as drawables.

Most of the functions in Xlib just add requests to an output buffer. These requests later execute asynchronously on the X server. Functions that return values of information stored in the server do not return (that is, they block) until an explicit reply is received or an error occurs. You can provide an error handler, which will be called when the error is reported.

If a client does not want a request to execute asynchronously, it can follow the request with a call to XSync, which blocks until all previously buffered asynchronous events have been sent and acted on. As an important side effect, the output buffer in Xlib is always flushed by a call to any function that returns a value from the server or waits for input.

Many Xlib functions will return an integer resource ID, which allows you to refer to objects stored on the X server. These can be of type Window, Font, Pixmap, Colormap, Cursor, and GContext, as defined in the file <X11/X.h>. These resources are created by requests and are destroyed (or freed) by requests or when connections are closed. Most of these resources are potentially sharable between applications, and in fact, windows are manipulated explicitly by window manager programs. Fonts and cursors are shared automatically across multiple screens. Fonts are loaded and unloaded as needed and are shared by multiple clients. Fonts are often cached in the server. Xlib provides no support for sharing graphics contexts between applications.

Client programs are informed of events. Events may either be side effects of a request (for example, restacking windows generates Expose events) or completely asynchronous (for example, from the keyboard). A client program asks to be informed of events. Because other applications can send events to your application, programs must be prepared to handle (or ignore) events of all types.

Input events (for example, a key pressed or the pointer moved) arrive asynchronously from the server and are queued until they are requested by an explicit call (for example, XNextEvent or XWindowEvent). In addition, some library functions (for example, XRaiseWindow) generate Expose and ConfigureRequest events. These events also arrive asynchronously, but the client may wish to explicitly wait for them by calling XSync after calling a function that can cause the server to generate events.

1.2. Errors

Some functions return Status, an integer error indication. If the function fails, it returns a zero. If the function returns a status of zero, it has not updated the return arguments. Because C does not provide multiple return values, many functions must return their results by writing
into client-passed storage. By default, errors are handled either by a standard library function or by one that you provide. Functions that return pointers to strings return NULL pointers if the string does not exist.

The X server reports protocol errors at the time that it detects them. If more than one error could be generated for a given request, the server can report any of them.

Because Xlib usually does not transmit requests to the server immediately (that is, it buffers them), errors can be reported much later than they actually occur. For debugging purposes, however, Xlib provides a mechanism for forcing synchronous behavior (see section 11.8.1). When synchronization is enabled, errors are reported as they are generated.

When Xlib detects an error, it calls an error handler, which your program can provide. If you do not provide an error handler, the error is printed, and your program terminates.

1.3. Standard Header Files

The following include files are part of the Xlib standard.

<X11/Xlib.h>

This is the main header file for Xlib. The majority of all Xlib symbols are declared by including this file. This file also contains the preprocessor symbol 
XlibSpecificationRelease. This symbol is defined to have the "5" in this release of the standard. (Earlier releases of Xlib did not have this symbol.)

<X11/X.h>

This file declares types and constants for the X protocol that are to be used by applications. It is included automatically from <X11/Xlib.h>, so application code should never need to reference this file directly.

<X11/Xcms.h>

This file contains symbols for much of the color management facilities described in chapter 6. All functions, types, and symbols with the prefix "Xcms", plus the Color Conversion Contexts macros, are declared in this file. <X11/Xlib.h> must be included before including this file.

<X11/Xutil.h>

This file declares various functions, types, and symbols used for inter-client communication and application utility functions, described in chapters 14 and 16. <X11/Xlib.h> must be included before including this file.

<X11/Xresource.h>

This file declares all functions, types, and symbols for the resource manager facilities, described in chapter 15. <X11/Xlib.h> must be included before including this file.

<X11/Xatom.h>

This file declares all predefined atoms, symbols with prefix "XA_".

<X11/cursorfont.h>

This file declares the cursor symbols for the standard cursor font, listed in appendix B. All symbols have the prefix "XC_".

<X11/keysymdef.h>

This file declares all standard KeySym values, symbols with prefix "XK_". The KeySyms are arranged in groups, and a preprocessor symbol controls inclusion of each group. The preprocessor symbol must be defined prior to inclusion of the file to obtain the associated values. The preprocessor symbols are: XK_MISC ALLANY, XK_LATIN1, XK_LATIN2, XK_LATIN3, XK_LATIN4, XK_KATAKANA, XK_ARABIC, XK_CYRILLIC, XK_GREEK, XK_TECHNICAL, XK_SPECIAL, XK_PUBLISHING, XK_APL, and XK_HEBREW.
<X11/keysym.h>

This file defines the preprocessor symbols XK_MISCELLANY, XK_LATIN1, XK_LATIN2, XK_LATIN3, XK_LATIN4, and XK_GREEK, and then includes <X11/keysymdef.h>.

<X11/Xlibint.h>

This file declares all the functions, types, and symbols used for extensions, described in appendix C. This file automatically includes <X11/Xlib.h>.

<X11/Xproto.h>

This file declares types and symbols for the basic X protocol, for use in implementing extensions. It is included automatically from <X11/Xlibint.h>, so application and extension code should never need to reference this file directly.

<X11/Xprotostr.h>

This file declares types and symbols for the basic X protocol, for use in implementing extensions. It is included automatically from <X11/Xproto.h>, so application and extension code should never need to reference this file directly.

<X11/X10.h>

This file declares all the functions, types, and symbols used for the X10 compatibility functions, described in appendix D.

1.4. Generic Values and Types

The following symbols are defined by Xlib and used throughout the manual:

- Xlib defines the type bool and the boolean values True and False.
- None is the universal null resource ID or atom.
- The type XID is used for generic resource IDs.
- The type XPointer is defined to be "char *" and is used as a generic opaque pointer to data.

1.5. Naming and Argument Conventions within Xlib

Xlib follows a number of conventions for the naming and syntax of the functions. Given that you remember what information the function requires, these conventions are intended to make the syntax of the functions more predictable.

The major naming conventions are:

- To differentiate the X symbols from the other symbols, the library uses mixed case for external symbols. It leaves lowercase for variables and all uppercase for user macros, as per existing convention.
- All Xlib functions begin with a capital X.
- The beginnings of all function names and symbols are capitalized.
- All user-visible data structures begin with a capital X. More generally, anything that a user might dereference begins with a capital X.
- Macros and other symbols do not begin with a capital X. To distinguish them from all user symbols, each word in the macro is capitalized.
- All elements of or variables in a data structure are in lowercase. Compound words, where needed, are constructed with underscores (_).
- The display argument, where used, is always first in the argument list.
- All resource objects, where used, occur at the beginning of the argument list immediately after the display argument.
• When a graphics context is present together with another type of resource (most commonly, a drawable), the graphics context occurs in the argument list after the other resource. Drawables outrank all other resources.
• Source arguments always precede the destination arguments in the argument list.
• The x argument always precedes the y argument in the argument list.
• The width argument always precedes the height argument in the argument list.
• Where the x, y, width, and height arguments are used together, the x and y arguments always precede the width and height arguments.
• Where a mask is accompanied with a structure, the mask always precedes the pointer to the structure in the argument list.

1.6. Programming Considerations
The major programming considerations are:
• Coordinates and sizes in X are actually 16-bit quantities. This decision was taken to minimize the bandwidth required for a given level of performance. Coordinates usually are declared as an "int" in the interface. Values larger than 16 bits are truncated silently. Sizes (width and height) are declared as unsigned quantities.
• Keyboards are the greatest variable between different manufacturers' workstations. If you want your program to be portable, you should be particularly conservative here.
• Many display systems have limited amounts of off-screen memory. If you can, you should minimize use of pixmaps and backing store.
• The user should have control of his screen real estate. Therefore, you should write your applications to react to window management rather than presume control of the entire screen. What you do inside of your top-level window, however, is up to your application. For further information, see chapter 14 and the Inter-Client Communication Conventions Manual.

1.7. Character Sets and Encodings
Some of the Xlib functions make reference to specific character sets and character encodings. The following ones are the most common:

X Portable Character Set
A basic set of 97 characters which are assumed to exist in all locales supported by Xlib. This set contains the following characters:

a..z A..Z 0..9
!"#$%&'()*+,-./:;<=>?@NV 0 T
<space>, <tab>, and <newline>

This is the left/lower half of the graphic character set of ISO8859-1 plus <space>, <tab>, and <newline>. It is also the set of graphic characters in 7-bit ASCII plus the same three control characters. The actual encoding of these characters on the host is system dependent.

Host Portable Character Encoding
The encoding of the X Portable Character Set on the host. The encoding itself is not defined by this standard, but the encoding must be the same in all locales supported by Xlib on the host. If a string is said to be in the Host Portable Character Encoding, then it only contains characters from the X Portable Character Set, in the host encoding.

Latin-1
The coded character set defined by the ISO8859-1 standard.
STRING encoding

Latin-1, plus tab and newline.

POSIX Portable Filename Character Set

The set of 65 characters which can be used in naming files on a POSIX-compliant host that are correctly processed in all locales. The set is:

a..z A..Z 0..9 \-

1.8. Formatting Conventions

Xlib – C Language X Interface uses the following conventions:

- Global symbols are printed in this special font. These can be either function names, symbols defined in include files, or structure names. Arguments are printed in italics.
- Each function is introduced by a general discussion that distinguishes it from other functions. The function declaration itself follows, and each argument is specifically explained. Although ANSI C function prototype syntax is not used, Xlib header files normally declare functions using function prototypes in ANSI C environments. General discussion of the function, if any is required, follows the arguments. Where applicable, the last paragraph of the explanation lists the possible Xlib error codes that the function can generate. For a complete discussion of the Xlib error codes, see section 11.8.2.
- To eliminate any ambiguity between those arguments that you pass and those that a function returns to you, the explanations for all arguments that you pass start with the word specifies or, in the case of multiple arguments, the word specify. The explanations for all arguments that are returned to you start with the word returns or, in the case of multiple arguments, the word return. The explanations for all arguments that you can pass and are returned start with the words specifies and returns.
- Any pointer to a structure that is used to return a value is designated as such by the _return suffix as part of its name. All other pointers passed to these functions are used for reading only. A few arguments use pointers to structures that are used for both input and output and are indicated by using the _in_out suffix.
Chapter 2

Display Functions

Before your program can use a display, you must establish a connection to the X server. Once you have established a connection, you then can use the Xlib macros and functions discussed in this chapter to return information about the display. This chapter discusses how to:

- Open (connect to) the display
- Obtain information about the display, image format, and screen
- Free client-created data
- Close (disconnect from) a display

The chapter concludes with a general discussion of what occurs when the connection to the X server is closed.

2.1. Opening the Display

To open a connection to the X server that controls a display, use `XOpenDisplay`.

```c
Display *XOpenDisplay (display_name)
char *display_name;
```

`display_name` Specifies the hardware display name, which determines the display and communications domain to be used. On a POSIX-conformant system, if the display_name is NULL, it defaults to the value of the DISPLAY environment variable.

The encoding and interpretation of the display name is implementation dependent. Strings in the Host Portable Character Encoding are supported; support for other characters is implementation dependent. On POSIX-conformant systems, the display name or DISPLAY environment variable can be a string in the format:

```
hostname:number.screen_number
```

- `hostname` Specifies the name of the host machine on which the display is physically attached. You follow the hostname with either a single colon (:) or a double colon (::).
- `number` Specifies the number of the display server on that host machine. You may optionally follow this display number with a period (.). A single CPU can have more than one display. Multiple displays are usually numbered starting with zero.
- `screen_number` Specifies the screen to be used on that server. Multiple screens can be controlled by a single X server. The screen_number sets an internal variable that can be accessed by using the DefaultScreen macro or the XDefaultScreen function if you are using languages other than C (see section 2.2.1).

For example, the following would specify screen 1 of display 0 on the machine named “dual-headed”:

```
dual-headed:0.1
```

The XOpenDisplay function returns a Display structure that serves as the connection to the X server and that contains all the information about that X server. XOpenDisplay connects your application to the X server through TCP or DECnet communications protocols, or through
some local inter-process communication protocol. If the hostname is a host machine name and a single colon (:) separates the hostname and display number, XOpenDisplay connects using TCP streams. If the hostname is not specified, Xlib uses whatever it believes is the fastest transport. If the hostname is a host machine name and a double colon (::) separates the hostname and display number, XOpenDisplay connects using DECnet. A single X server can support any or all of these transport mechanisms simultaneously. A particular Xlib implementation can support many more of these transport mechanisms.

If successful, XOpenDisplay returns a pointer to a Display structure, which is defined in <X11/Xlib.h>. If XOpenDisplay does not succeed, it returns NULL. After a successful call to XOpenDisplay, all of the screens in the display can be used by the client. The screen number specified in the display_name argument is returned by the DefaultScreen macro (or the XDefaultScreen function). You can access elements of the Display and Screen structures only by using the information macros or functions. For information about using macros and functions to obtain information from the Display structure, see section 2.2.1.

X servers may implement various types of access control mechanisms (see section 9.8).

2.2. Obtaining Information about the Display, Image Formats, or Screens
The Xlib library provides a number of useful macros and corresponding functions that return data from the Display structure. The macros are used for C programming, and their corresponding function equivalents are for other language bindings. This section discusses the:

- Display macros
- Image format macros
- Screen macros

All other members of the Display structure (that is, those for which no macros are defined) are private to Xlib and must not be used. Applications must never directly modify or inspect these private members of the Display structure.

Note
The XDisplayWidth, XDisplayHeight, XDisplayCells, XDisplayPlanes, XDisplayWidthMM, and XDisplayHeightMM functions in the next sections are misnamed. These functions really should be named Screenwhatever and XScreenwhatever, not Displaywhatever or XDisplaywhatever. Our apologies for the resulting confusion.

2.2.1. Display Macros
Applications should not directly modify any part of the Display and Screen structures. The members should be considered read-only, although they may change as the result of other operations on the display.

The following lists the C language macros, their corresponding function equivalents that are for other language bindings, and what data they both can return.

AllPlanes

unsigned long XAllPlanes()

Both return a value with all bits set to 1 suitable for use in a plane argument to a procedure.

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

BlackPixel(display, screen_number)

unsigned long XBlackPixel(display, screen_number)
    Display *display;
    int screen_number;

display Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the black pixel value for the specified screen.

WhitePixel(display, screen_number)

unsigned long XWhitePixel(display, screen_number)
    Display *display;
    int screen_number;

display Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the white pixel value for the specified screen.

ConnectionNumber(display)

int XConnectionNumber(display)
    Display *display;

display Specifies the connection to the X server.
Both return a connection number for the specified display. On a POSIX-conformant system,
this is the file descriptor of the connection.

DefaultColormap(display, screen_number)

Colormap XDefaultColormap(display, screen_number)
    Display *display;
    int screen_number;

display Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the default colormap ID for allocation on the specified screen. Most routine allo-
cations of color should be made out of this colormap.

DefaultDepth(display, screen_number)

int XDefaultDepth(display, screen_number)
    Display *display;
    int screen_number;

display Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the depth (number of planes) of the default root window for the specified screen. Other depths may also be supported on this screen (see XMatchVisualInfo).

To determine the number of depths that are available on a given screen, use XListDepths.

```c
int *XListDepths(display, screen_number, count_return)
    Display *display;
    int screen_number;
    int *count_return;

    display Specifies the connection to the X server.
    screen_number Specifies the appropriate screen number on the host server.
    count_return Returns the number of depths.
```

The XListDepths function returns the array of depths that are available on the specified screen. If the specified screen_number is valid and sufficient memory for the array can be allocated, XListDepths sets count_return to the number of available depths. Otherwise, it does not set count_return and returns NULL. To release the memory allocated for the array of depths, use XFree.

DefaultGC(display, screen_number)

```c
GC XDefaultGC(display, screen_number)
    Display *display;
    int screen_number;

    display Specifies the connection to the X server.
    screen_number Specifies the appropriate screen number on the host server.
```

Both return the default graphics context for the root window of the specified screen. This GC is created for the convenience of simple applications and contains the default GC components with the foreground and background pixel values initialized to the black and white pixels for the screen, respectively. You can modify its contents freely because it is not used in any Xlib function. This GC should never be freed.

DefaultRootWindow(display)

```c
Window XDefaultRootWindow(display)
    Display *display;

    display Specifies the connection to the X server.
```

Both return the root window for the default screen.

DefaultScreenOfDisplay(display)

```c
Screen *XDefaultScreenOfDisplay(display)
    Display *display;

    display Specifies the connection to the X server.
```

Both return a pointer to the default screen.
ScreenOfDisplay(display, screen_number)

Screen *XScreenOfDisplay(display, screen_number)
    Display *display;
    int screen_number;

display  Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return a pointer to the indicated screen.

DefaultScreen(display)

int XDefaultScreen(display)
    Display *display;

display  Specifies the connection to the X server.
Both return the default screen number referenced by the XOpenDisplay function. This macro
or function should be used to retrieve the screen number in applications that will use only a
single screen.

DefaultVisual(display, screen_number)

Visual *XDefaultVisual(display, screen_number)
    Display *display;
    int screen_number;

display  Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the default visual type for the specified screen. For further information about
visual types, see section 3.1.

DisplayCells(display, screen_number)

int XDisplayCells(display, screen_number)
    Display *display;
    int screen_number;

display  Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the number of entries in the default colormap.

DisplayPlanes(display, screen_number)

int XDisplayPlanes(display, screen_number)
    Display *display;
    int screen_number;

display  Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the depth of the root window of the specified screen. For an explanation of depth,
see the glossary.
DisplayString(display)

char *XDisplayString(display)
Display *display;

display Specifies the connection to the X server.
Both return the string that was passed to XOpenDisplay when the current display was opened. On POSIX-conformant systems, if the passed string was NULL, these return the value of the DISPLAY environment variable when the current display was opened. These are useful to applications that invoke the fork system call and want to open a new connection to the same display from the child process as well as for printing error messages.

long XMaxRequestSize(display)
Display *display;

display Specifies the connection to the X server.
XMaxRequestSize returns the maximum request size (in 4-byte units) supported by the server. Single protocol requests to the server can be no longer than this size. The protocol guarantees the size to be no smaller than 4096 units (16384 bytes). Xlib automatically breaks data up into multiple protocol requests as necessary for the following functions: XDrawPoints, XDrawRectangles, XDrawSegments, XFillAres, XFillRectangles, and XPutImage.

LastKnownRequestProcessed(display)

unsigned long XLastKnownRequestProcessed(display)
Display *display;

display Specifies the connection to the X server.
Both extract the full serial number of the last request known by Xlib to have been processed by the X server. Xlib automatically sets this number when replies, events, and errors are received.

NextRequest(display)

unsigned long XNextRequest(display)
Display *display;

display Specifies the connection to the X server.
Both extract the full serial number that is to be used for the next request. Serial numbers are maintained separately for each display connection.

ProtocolVersion(display)

int XProtocolVersion(display)
Display *display;

display Specifies the connection to the X server.
Both return the major version number (11) of the X protocol associated with the connected display.
ProtocolRevision(display)

int XProtocolRevision(display)
    Display *display;

display Specifies the connection to the X server.
Both return the minor protocol revision number of the X server.

QLength(display)

int XQLength(display)
    Display *display;

display Specifies the connection to the X server.
Both return the length of the event queue for the connected display. Note that there may be
more events that have not been read into the queue yet (see XEventsQueued).

RootWindow(display, screen_number)

Window XRootWindow(display, screen_number)
    Display *display;
    int screen_number;

display Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the root window. These are useful with functions that need a drawable of a partic¬
ular screen and for creating top-level windows.

ScreenCount(display)

int XScreenCount(display)
    Display *display;

display Specifies the connection to the X server.
Both return the number of available screens.

ServerVendor(display)

char *XServerVendor(display)
    Display *display;

display Specifies the connection to the X server.
Both return a pointer to a null-terminated string that provides some identification of the owner
of the X server implementation. If the data returned by the server is in the Latin Portable
Character Encoding, then the string is in the Host Portable Character Encoding. Otherwise, the
contents of the string are implementation dependent.

VendorRelease(display)

int XVendorRelease(display)
    Display *display;
display Specifies the connection to the X server.
Both return a number related to a vendor’s release of the X server.

2.2.2. Image Format Functions and Macros
Applications are required to present data to the X server in a format that the server demands.
To help simplify applications, most of the work required to convert the data is provided by
Xlib (see sections 8.7 and 16.8).
The XPixmapFormatValues structure provides an interface to the pixmap format information
that is returned at the time of a connection setup. It contains:

typedef struct {
    int depth;
    int bits_per_pixel;
    int scanline_pad;
} XPixmapFormatValues;

To obtain the pixmap format information for a given display, use XListPixmapFormats.
XPixmapFormatValues *XListPixmapFormats(display, count_return)
Display *display;
int *count_return;

display Specifies the connection to the X server.
count_return Returns the number of pixmap formats that are supported by the display.
The XListPixmapFormats function returns an array of XPixmapFormatValues structures
that describe the types of Z format images supported by the specified display. If insufficient
memory is available, XListPixmapFormats returns NULL. To free the allocated storage for
the XPixmapFormatValues structures, use XFree.
The following lists the C language macros, their corresponding function equivalents that are for
other language bindings, and what data they both return for the specified server and screen.
These are often used by toolkits as well as by simple applications.

ImageByteOrder(display)

int XImageByteOrder(display)
Display *display;

display Specifies the connection to the X server.
Both specify the required byte order for images for each scanline unit in XY format (bitmap)
or for each pixel value in Z format. The macro or function can return either LSBFirst or
MSBFirst.

BitmapUnit(display)

int XBitmapUnit(display)
Display *display;

display Specifies the connection to the X server.
Both return the size of a bitmap’s scanline unit in bits. The scanline is calculated in multiples
of this value.
BitmapBitOrder(display)

int XBitmapBitOrder(display)
    Display *display;

display     Specifies the connection to the X server.
Within each bitmap unit, the left-most bit in the bitmap as displayed on the screen is either the
least-significant or most-significant bit in the unit. This macro or function can return
LSBFirst or MSBFirst.

BitmapPad(display)

int XBitmapPad(display)
    Display *display;

display     Specifies the connection to the X server.
Each scanline must be padded to a multiple of bits returned by this macro or function.

DisplayHeight(display, screen_number)

int XDisplayHeight(display, screen_number)
    Display *display;
    int screen_number;

display     Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return an integer that describes the height of the screen in pixels.

DisplayHeightMM(display, screen_number)

int XDisplayHeightMM(display, screen_number)
    Display *display;
    int screen_number;

display     Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the height of the specified screen in millimeters.

DisplayWidth(display, screen_number)

int XDisplayWidth(display, screen_number)
    Display *display;
    int screen_number;

display     Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the width of the screen in pixels.
DisplayWidthMM(display, screen_number)

int XDisplayWidthMM(display, screen_number)

Display *display;
int screen_number;

display Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
Both return the width of the specified screen in millimeters.

2.2.3. Screen Information Macros
The following lists the C language macros, their corresponding function equivalents that are for other language bindings, and what data they both can return. These macros or functions all take a pointer to the appropriate screen structure.

BlackPixelOfScreen(screen)

unsigned long XBlackPixelOfScreen(screen)
Screen *screen;

screen Specifies the appropriate Screen structure.
Both return the black pixel value of the specified screen.

WhitePixelOfScreen(screen)

unsigned long XWhitePixelOfScreen(screen)
Screen *screen;

screen Specifies the appropriate Screen structure.
Both return the white pixel value of the specified screen.

CellsOfScreen(screen)

int XCellsOfScreen(screen)
Screen *screen;

screen Specifies the appropriate Screen structure.
Both return the number of colormap cells in the default colormap of the specified screen.

DefaultColormapOfScreen(screen)

Colormap XDefaultColormapOfScreen(screen)
Screen *screen;

screen Specifies the appropriate Screen structure.
Both return the default colormap of the specified screen.

DefaultDepthOfScreen(screen)

int XDefaultDepthOfScreen(screen)
Screen *screen;
screen Specifies the appropriate Screen structure.
Both return the depth of the root window.

DefaultGCOfScreen(screen)

GC XDefaultGCOfScreen(screen)
    Screen *screen;

screen Specifies the appropriate Screen structure.
Both return a default graphics context (GC) of the specified screen, which has the same depth
as the root window of the screen. The GC must never be freed.

DefaultVisualOfScreen(screen)

Visual *XDefaultVisualOfScreen(screen)
    Screen *screen;

screen Specifies the appropriate Screen structure.
Both return the default visual of the specified screen. For information on visual types, see sec¬
tion 3.1.

DoesBackingStore(screen)

int XDoesBackingStore(screen)
    Screen *screen;

screen Specifies the appropriate Screen structure.
Both return a value indicating whether the screen supports backing stores. The value returned
can be one of WhenMapped, NotUseful, or Always (see section 3.2.4).

DoesSaveUnders(screen)

Bool XDoesSaveUnders(screen)
    Screen *screen;

screen Specifies the appropriate Screen structure.
Both return a Boolean value indicating whether the screen supports save unders. If True, the
screen supports save unders. If False, the screen does not support save unders (see section
3.2.5).

DisplayOfScreen(screen)

Display *XDisplayOfScreen(screen)
    Screen *screen;

screen Specifies the appropriate Screen structure.
Both return the display of the specified screen.

int XScreenNumberOfScreen(screen)
    Screen *screen;
screen  Specifies the appropriate Screen structure.
The XScreenNumberOfScreen function returns the screen index number of the specified screen.

EventMaskOfScreen(screen)

long XEventMaskOfScreen(screen)
    Screen *screen;

screen  Specifies the appropriate Screen structure.
Both return the event mask of the root window for the specified screen at connection setup time.

WidthOfScreen(screen)

int XWidthOfScreen(screen)
    Screen *screen;

screen  Specifies the appropriate Screen structure.
Both return the width of the specified screen in pixels.

HeightOfScreen(screen)

int XHeightOfScreen(screen)
    Screen *screen;

screen  Specifies the appropriate Screen structure.
Both return the height of the specified screen in pixels.

WidthMMOfScreen(screen)

int XWidthMMOfScreen(screen)
    Screen *screen;

screen  Specifies the appropriate Screen structure.
Both return the width of the specified screen in millimeters.

HeightMMOfScreen(screen)

int XHeightMMOfScreen(screen)
    Screen *screen;

screen  Specifies the appropriate Screen structure.
Both return the height of the specified screen in millimeters.

MaxCmapsOfScreen(screen)

int XMaxCmapsOfScreen(screen)
    Screen *screen;

screen  Specifies the appropriate Screen structure.
Both return the maximum number of installed colormaps supported by the specified screen (see section 9.3).

\[ \text{MinCmapsOfScreen}(\text{screen}) \]

\[
\text{int XMinCmapsOfScreen}(\text{screen})
\]

\[
\text{Screen *screen;}
\]

\[
\text{screen} \quad \text{Specifies the appropriate Screen structure.}
\]

Both return the minimum number of installed colormaps supported by the specified screen (see section 9.3).

\[ \text{PlanesOfScreen}(\text{screen}) \]

\[
\text{int XPlanesOfScreen}(\text{screen})
\]

\[
\text{Screen *screen;}
\]

\[
\text{screen} \quad \text{Specifies the appropriate Screen structure.}
\]

Both return the depth of the root window.

\[ \text{RootWindowOfScreen}(\text{screen}) \]

\[
\text{Window XRoodWindowOfScreen}(\text{screen})
\]

\[
\text{Screen *screen;}
\]

\[
\text{screen} \quad \text{Specifies the appropriate Screen structure.}
\]

Both return the root window of the specified screen.

2.3. Generating a NoOperation Protocol Request

To execute a NoOperation protocol request, use XNoOp.

\[ \text{XNoOp}(\text{display}) \]

\[
\text{Display *display;}
\]

\[
\text{display} \quad \text{Specifies the connection to the X server.}
\]

The XNoOp function sends a NoOperation protocol request to the X server, thereby exercising the connection.

2.4. Freeing Client-Created Data

To free in-memory data that was created by an Xlib function, use XFree.

\[ \text{XFree}(\text{data}) \]

\[
\text{void *data;}
\]

\[
\text{data} \quad \text{Specifies the data that is to be freed.}
\]

The XFree function is a general-purpose Xlib routine that frees the specified data. You must use it to free any objects that were allocated by Xlib, unless an alternate function is explicitly specified for the object.

2.5. Closing the Display

To close a display or disconnect from the X server, use XCloseDisplay.
XCloseDisplay(display)
Display *display;

display Specifies the connection to the X server.
The XCloseDisplay function closes the connection to the X server for the display specified in
the Display structure and destroys all windows, resource IDs (Window, Font, Pixmap, Colormap, Cursor, and GContext), or other resources that the client has created on this
display, unless the close-down mode of the resource has been changed (see XSetCloseDown-
Mode). Therefore, these windows, resource IDs, and other resources should never be refer¬
enced again or an error will be generated. Before exiting, you should call XCloseDisplay
explicitly so that any pending errors are reported as XCloseDisplay performs a final XSync
operation.

XCloseDisplay can generate a BadGC error.

Xlib provides a function to permit the resources owned by a client to survive after the client’s
connection is closed. To change a client’s close-down mode, use XSetCloseDownMode.
XSetCloseDownMode(display, close_mode)
Display *display;
int close_mode;

display Specifies the connection to the X server.
close_mode Specifies the client close-down mode. You can pass DestroyAll, RetainPer-
manent, or RetainTemporary.

The XSetCloseDownMode defines what will happen to the client’s resources at connection
close. A connection starts in DestroyAll mode. For information on what happens to the
client’s resources when the close_mode argument is RetainPermanent or RetainTemporary,
see section 2.6.

XSetCloseDownMode can generate a BadValue error.

2.6. X Server Connection Close Operations

When the X server’s connection to a client is closed either by an explicit call to
XCloseDisplay or by a process that exits, the X server performs the following automatic
operations:
• It disowns all selections owned by the client (see XSetSelectionOwner).
• It performs an XUngrabPointer and XUngrabKeyboard if the client has actively
grabbed the pointer or the keyboard.
• It performs an XUngrabServer if the client has grabbed the server.
• It releases all passive grabs made by the client.
• It marks all resources (including colormap entries) allocated by the client either as per-
manent or temporary, depending on whether the close-down mode is RetainPermanent
or RetainTemporary. However, this does not prevent other client applications from
explicitly destroying the resources (see XSetCloseDownMode).

When the close-down mode is DestroyAll, the X server destroys all of a client’s resources as
follows:
• It examines each window in the client’s save-set to determine if it is an inferior (subwin-
dow) of a window created by the client. (The save-set is a list of other clients’ win-
dows, which are referred to as save-set windows.) If so, the X server reparents the save-
set window to the closest ancestor so that the save-set window is 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.
- It performs a `MapWindow` request on the save-set window if the save-set window is unmapped. The X server does this even if the save-set window was not an inferior of a window created by the client.
- It destroys all windows created by the client.
- It performs the appropriate free request on each nonwindow resource created by the client in the server (for example, `Font`, `Pixmap`, `Cursor`, `Colormap`, and `GContext`).
- It frees all colors and colormap entries allocated by a client application.

Additional processing occurs when the last connection to the X server closes. An X server goes through a cycle of having no connections and having some connections. When the last connection to the X server closes as a result of a connection closing with the close_mode of `DestroyAll`, the X server does the following:
- It resets its state as if it had just been started. The X server begins by destroying all lingering resources from clients that have terminated in `RetainPermanent` or `RetainTemporary` mode.
- It deletes all but the predefined atom identifiers.
- It deletes all properties on all root windows (see section 4.3).
- It resets all device maps and attributes (for example, key click, bell volume, and acceleration) as well as the access control list.
- It restores the standard root tiles and cursors.
- It restores the default font path.
- It restores the input focus to state `PointerRoot`.

However, the X server does not reset if you close a connection with a close-down mode set to `RetainPermanent` or `RetainTemporary`. 
Chapter 3

Window Functions

In the X Window System, a window is a rectangular area on the screen that lets you view graphic output. Client applications can display overlapping and nested windows on one or more screens that are driven by X servers on one or more machines. Clients who want to create windows must first connect their program to the X server by calling XOpenDisplay. This chapter begins with a discussion of visual types and window attributes. The chapter continues with a discussion of the Xlib functions you can use to:

- Create windows
- Destroy windows
- Map windows
- Unmap windows
- Configure windows
- Change the stacking order
- Change window attributes

This chapter also identifies the window actions that may generate events.

Note that it is vital that your application conform to the established conventions for communicating with window managers for it to work well with the various window managers in use (see section 14.1). Toolkits generally adhere to these conventions for you, relieving you of the burden. Toolkits also often supersede many functions in this chapter with versions of their own. Refer to the documentation for the toolkit you are using for more information.

3.1. Visual Types

On some display hardware, it may be possible to deal with color resources in more than one way. For example, you may be able to deal with a screen of either 12-bit depth with arbitrary mapping of pixel to color (pseudo-color) or 24-bit depth with 8 bits of the pixel dedicated to each of red, green, and blue. These different ways of dealing with the visual aspects of the screen are called visuals. For each screen of the display, there may be a list of valid visual types supported at different depths of the screen. Because default windows and visual types are defined for each screen, most simple applications need not deal with this complexity. Xlib provides macros and functions that return the default root window, the default depth of the default root window, and the default visual type (see sections 2.2.1 and 16.7).

Xlib uses an opaque Visual structure that contains information about the possible color mapping. The visual utility functions (see section 16.7) use an XVisualInfo structure to return this information to an application. The members of this structure pertinent to this discussion are class, red_mask, green_mask, blue_mask, bits_per_rgb, and colormap_size. The class member specifies one of the possible visual classes of the screen and can be StaticGray, StaticColor, TrueColor, Grayscale, PseudoColor, or DirectColor.

The following concepts may serve to make the explanation of visual types clearer. The screen can be color or grayscale, can have a colormap that is writable or read-only, and can also have a colormap whose indices are decomposed into separate RGB pieces, provided one is not on a grayscale screen. This leads to the following diagram:
Conceptually, as each pixel is read out of video memory for display on the screen, it goes through a look-up stage by indexing into a colormap. Colormaps can be manipulated arbitrarily on some hardware, in limited ways on other hardware, and not at all on other hardware. The visual types affect the colormap and the RGB values in the following ways:

- **For PseudoColor**, a pixel value indexes a colormap to produce independent RGB values, and the RGB values can be changed dynamically.

- **Grayscale** is treated the same way as PseudoColor except that the primary that drives the screen is undefined. Thus, the client should always store the same value for red, green, and blue in the colormaps.

- **For DirectColor**, a pixel value is decomposed into separate RGB subfields, and each subfield separately indexes the colormap for the corresponding value. The RGB values can be changed dynamically.

- **TrueColor** is treated the same way as DirectColor except that the colormap has predefined, read-only RGB values. These RGB values are server-dependent but provide linear or near-linear ramps in each primary.

- **StaticColor** is treated the same way as PseudoColor except that the colormap has predefined, read-only, server-dependent RGB values.

- **StaticGray** is treated the same way as StaticColor except that the RGB values are equal for any single pixel value, thus resulting in shades of gray. StaticGray with a two-entry colormap can be thought of as monochrome.

The red_mask, green_mask, and blue_mask members are only defined for DirectColor and TrueColor. Each has one contiguous set of bits with no intersections. The bits_per_rgb member specifies the log base 2 of the number of distinct color values (individually) of red, green, and blue. Actual RGB values are unsigned 16-bit numbers. The colormap_size member defines the number of available colormap entries in a newly created colormap. For DirectColor and TrueColor, this is the size of an individual pixel subfield.

To obtain the visual ID from a Visual, use `XVisualIDFromVisual`.

```c
VisualID XVisualIDFromVisual(visual)
    Visual *visual;

visual        Specifies the visual type.
```

The `XVisualIDFromVisual` function returns the visual ID for the specified visual type.

### 3.2. Window Attributes

All InputOutput windows have a border width of zero or more pixels, an optional background, an event suppression mask (which suppresses propagation of events from children), and a property list (see section 4.3). The window border and background can be a solid color or a pattern, called a tile. All windows except the root have a parent and are clipped by their parent. If a window is stacked on top of another window, it obscures that other window for the purpose of input. If a window has a background (almost all do), it obscures the other window for purposes of output. Attempts to output to the obscured area do nothing, and no input
events (for example, pointer motion) are generated for the obscured area. Windows also have associated property lists (see section 4.3).

Both InputOutput and InputOnly windows have the following common attributes, which are the only attributes of an InputOnly window:

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

If you specify any other attributes for an InputOnly window, a BadMatch error results.

InputOnly windows are used for controlling input events in situations where InputOutput windows are unnecessary. InputOnly windows are invisible; can only be used to control such things as cursors, input event generation, and grabbing; and cannot be used in any graphics requests. Note that InputOnly windows cannot have InputOutput windows as inferiors.

Windows have borders of a programmable width and pattern as well as a background pattern or tile. Pixel values can be used for solid colors. The background and border pixmaps can be destroyed immediately after creating the window if no further explicit references to them are to be made. The pattern can either be relative to the parent or absolute. If ParentRelative, the parent's background is used.

When windows are first created, they are not visible (not mapped) on the screen. Any output to a window that is not visible on the screen and that does not have backing store will be discarded. An application may wish to create a window long before it is mapped to the screen. When a window is eventually mapped to the screen (using XMapWindow), the X server generates an Expose event for the window if backing store has not been maintained.

A window manager can override your choice of size, border width, and position for a top-level window. Your program must be prepared to use the actual size and position of the top window. It is not acceptable for a client application to resize itself unless in direct response to a human command to do so. Instead, either your program should use the space given to it, or if the space is too small for any useful work, your program might ask the user to resize the window. The border of your top-level window is considered fair game for window managers.

To set an attribute of a window, set the appropriate member of the XSetWindowAttributes structure and OR in the corresponding value bitmask in your subsequent calls to XCreateWindow and XChangeWindowAttributes, or use one of the other convenience functions that set the appropriate attribute. The symbols for the value mask bits and the XSetWindowAttributes structure are:

/* Window attribute value mask bits */

#define CWBackPixmap (1L<<0)
#define CWBackPixel (1L<<1)
#define CWBorderPixmap (1L<<2)
#define CWBorderPixel (1L<<3)
#define CWBitGravity (1L<<4)
#define CWWinGravity (1L<<5)
#define CWBackingStore (1L<<6)
#define CWBackingPlanes (1L<<7)
#define CWBackingPixel (1L<<8)
#define CWOverrideRedirect (1L<<9)
#define CWSaveUnder (1L<<10)
#define CWEventMask (1L<<11)
#define CWDontPropagate (1L<<12)
Xlib – C Library

#define CWColormap
#define CWCursor

/* Values */

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

The following lists the defaults for each window attribute and indicates whether the attribute is applicable to InputOutput and InputOnly windows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>InputOutput</th>
<th>InputOnly</th>
</tr>
</thead>
<tbody>
<tr>
<td>background-pixmap</td>
<td>None</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>background-pixel</td>
<td>Undefined</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>border-pixmap</td>
<td>CopyFromParent</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>border-pixel</td>
<td>Undefined</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>bit-gravity</td>
<td>ForgetGravity</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>win-gravity</td>
<td>NorthWestGravity</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>backing-store</td>
<td>NotUseful</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>backing-planes</td>
<td>All ones</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>backing-pixel</td>
<td>zero</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>save-under</td>
<td>False</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>event-mask</td>
<td>empty set</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>do-not-propagate-mask</td>
<td>empty set</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>override-redirect</td>
<td>False</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>colormap</td>
<td>CopyFromParent</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>cursor</td>
<td>None</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

3.2.1. Background Attribute

Only InputOutput windows can have a background. You can set the background of an InputOutput window by using a pixel or a pixmap.

The background-pixmap attribute of a window specifies the pixmap to be used for a window’s background. This pixmap can be of any size, although some sizes may be faster than others. The background-pixel attribute of a window specifies a pixel value used to paint a window’s background in a single color.

You can set the background-pixmap to a pixmap, None (default), or ParentRelative. You can set the background-pixel of a window to any pixel value (no default). If you specify a background-pixel, it overrides either the default background-pixmap or any value you may
have set in the background-pixmap. A pixmap of an undefined size that is filled with the background-pixel is used for the background. Range checking is not performed on the background pixel; it simply is truncated to the appropriate number of bits.

If you set the background-pixmap, it overrides the default. The background-pixmap and the window must have the same depth, or a BadMatch error results. If you set background-pixmap to None, the window has no defined background. If you set the background-pixmap to ParentRelative:

- The parent window’s background-pixmap is used. The child window, however, must have the same depth as its parent, or a BadMatch error results.
- If the parent window has a background-pixmap of None, the window also has a background-pixmap of None.
- A copy of the parent window’s background-pixmap is not made. The parent’s background-pixmap is examined each time the child window’s background-pixmap is required.
- The background tile origin always aligns with the parent window’s background tile origin. If the background-pixmap is not ParentRelative, the background tile origin is the child window’s origin.

Setting a new background, whether by setting background-pixmap or background-pixel, overrides any previous background. The background-pixmap can be freed immediately if no further explicit reference is made to it (the X server will keep a copy to use when needed). If you later draw into the pixmap used for the background, what happens is undefined because the X implementation is free to make a copy of the pixmap or to use the same pixmap.

When no valid contents are available for regions of a window and either the regions are visible or the server is maintaining backing store, the server automatically tiles the regions with the window’s background unless the window has a background of None. If the background is None, the previous screen contents from other windows of the same depth as the window are simply left in place as long as 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. Expose events are then generated for the regions, even if the background-pixmap is None (see section 10.9).

3.2.2. Border Attribute

Only InputOutput windows can have a border. You can set the border of an InputOutput window by using a pixel or a pixmap.

The border-pixmap attribute of a window specifies the pixmap to be used for a window’s border. The border-pixel attribute of a window specifies a pixmap of undefined size filled with that pixel be used for a window’s border. Range checking is not performed on the background pixel; it simply is truncated to the appropriate number of bits. The border tile origin is always the same as the background tile origin.

You can also set the border-pixmap to a pixmap of any size (some may be faster than others) or to CopyFromParent (default). You can set the border-pixel to any pixel value (no default).

If you set a border-pixmap, it overrides the default. The border-pixmap and the window must have the same depth, or a BadMatch error results. If you set the border-pixmap to CopyFromParent, the parent window’s border-pixmap is copied. Subsequent changes to the parent window’s border attribute do not affect the child window. However, the child window must have the same depth as the parent window, or a BadMatch error results.

The border-pixmap can be freed immediately if no further explicit reference is made to it. If you later draw into the pixmap used for the border, what happens is undefined because the X implementation is free either to make a copy of the pixmap or to use the same pixmap. If you specify a border-pixel, it overrides either the default border-pixmap or any value you may have
set in the border-pixmap. All pixels in the window’s border will be set to the border-pixel. Setting a new border, whether by setting border-pixel or by setting border-pixmap, overrides any previous border.

Output to a window is always clipped to the inside of the window. Therefore, graphics operations never affect the window border.

### 3.2.3. Gravity Attributes

The bit gravity of a window defines which region of the window should be retained when an `InputOutput` window is resized. The default value for the bit-gravity attribute is `ForgetGravity`. The window gravity of a window allows you to define how the `InputOutput` or `InputOnly` window should be repositioned if its parent is resized. The default value for the win-gravity attribute is `NorthWestGravity`.

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

<table>
<thead>
<tr>
<th>Gravity Direction</th>
<th>Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>NorthWestGravity</td>
<td>((0, 0))</td>
</tr>
<tr>
<td>NorthGravity</td>
<td>((\text{Width}/2, 0))</td>
</tr>
<tr>
<td>NorthEastGravity</td>
<td>((\text{Width}, 0))</td>
</tr>
<tr>
<td>WestGravity</td>
<td>((0, \text{Height}/2))</td>
</tr>
<tr>
<td>CenterGravity</td>
<td>((\text{Width}/2, \text{Height}/2))</td>
</tr>
<tr>
<td>EastGravity</td>
<td>((\text{Width}, \text{Height}/2))</td>
</tr>
<tr>
<td>SouthWestGravity</td>
<td>((0, \text{Height}))</td>
</tr>
<tr>
<td>SouthGravity</td>
<td>((\text{Width}/2, \text{Height}))</td>
</tr>
<tr>
<td>SouthEastGravity</td>
<td>((\text{Width}, \text{Height}))</td>
</tr>
</tbody>
</table>

When a window with one of these bit-gravity values is resized, the corresponding pair defines the change in position of each pixel in the window. When a window with one of these win-gravities has its parent window resized, the corresponding pair defines the change in position of the window within the parent. When a window is so repositioned, a GravityNotify event is generated (see section 10.10.5).

A bit-gravity of `StaticGravity` indicates that the contents or origin should not move relative to the origin of the root window. If the change in size of the window is coupled with a change in position \((x, y)\), then for bit-gravity the change in position of each pixel is \((-x, -y)\), and for win-gravity the change in position of a child when its parent is so resized is \((-x, -y)\). Note that `StaticGravity` still only takes effect when the width or height of the window is changed, not when the window is moved.

A bit-gravity of `ForgetGravity` indicates that the window’s contents are always discarded after a size change, even if a backing store or save under has been requested. The window is tiled with its background and zero or more `Expose` events are generated. If no background is defined, the existing screen contents are not altered. Some X servers may also ignore the specified bit-gravity and always generate `Expose` events.

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 `UnmapGravity` is like `NorthWestGravity` (the window is not moved), except the child is also unmapped when the parent is resized, and an UnmapNotify event is generated.
3.2.4. Backing Store Attribute

Some implementations of the X server may choose to maintain the contents of InputOutput windows. If the X server maintains the contents of a window, the off-screen saved pixels are known as backing store. The backing store advises the X server on what to do with the contents of a window. The backing-store attribute can be set to NotUseful (default), When-Mapped, or Always.

A backing-store attribute of NotUseful advises the X server that maintaining contents is unnecessary, although some X implementations may still choose to maintain contents and, therefore, not generate Expose events. A backing-store attribute of WhenMapped advises the X server that maintaining contents of obscured regions when the window is mapped would be beneficial. In this case, the server may generate an Expose event when the window is created. A backing-store attribute of Always advises the X server that maintaining contents even when the window is unmapped would be beneficial. Even if the window is larger than its parent, this is a request to the X server to maintain complete contents, not just the region within the parent window boundaries. While the X server maintains the window’s contents, Expose events normally are not generated, but the X server may stop maintaining contents at any time.

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

3.2.5. Save Under Flag

Some server implementations may preserve contents of InputOutput windows under other InputOutput windows. This is not the same as preserving the contents of a window for you. You may get better visual appeal if transient windows (for example, pop-up menus) request that the system preserve the screen contents under them, so the temporarily obscured applications do not have to repaint.

You can set the save-under flag to True or False (default). If save-under is True, the X server is advised that, when this window is mapped, saving the contents of windows it obscures would be beneficial.

3.2.6. Backing Planes and Backing Pixel Attributes

You can set backing planes to indicate (with bits set to 1) which bit planes of an InputOutput window hold dynamic data that must be preserved in backing store and during save unders. The default value for the backing-planes attribute is all bits set to 1. You can set backing pixel to specify what bits to use in planes not covered by backing planes. The default value for the backing-pixel attribute is all bits set to 0. The X server is free to save only the specified bit planes in the backing store or the save under and is free to regenerate the remaining planes with the specified pixel value. Any extraneous bits in these values (that is, those bits beyond the specified depth of the window) may be simply ignored. If you request backing store or save unders, you should use these members to minimize the amount of off-screen memory required to store your window.

3.2.7. Event Mask and Do Not Propagate Mask Attributes

The event mask defines which events the client is interested in for this InputOutput or InputOnly window (or, for some event types, inferiors of this window). The event mask is the bitwise inclusive OR of zero or more of the valid event mask bits. You can specify that no maskable events are reported by setting NoEventMask (default).

The do-not-propagate-mask attribute defines which events should not be propagated to ancestor windows when no client has the event type selected in this InputOutput or InputOnly window. The do-not-propagate-mask is the bitwise inclusive OR of zero or more of the following masks: KeyPress, KeyRelease, ButtonPress, ButtonRelease, PointerMotion, Button1Motion, Button2Motion, Button3Motion, Button4Motion, Button5Motion, and
ButtonMotion. You can specify that all events are propagated by setting NoEventMask (default).

3.2.8. Override Redirect Flag
To control window placement or to add decoration, a window manager often needs to intercept (redirect) any map or configure request. Pop-up windows, however, often need to be mapped without a window manager getting in the way. To control whether an InputOutput or InputOnly window is to ignore these structure control facilities, use the override-redirect flag.

The override-redirect flag specifies whether map and configure requests on this window should override a SubstructureRedirectMask on the parent. You can set the override-redirect flag to True or False (default). Window managers use this information to avoid tampering with pop-up windows (see also chapter 14).

3.2.9. Colormap Attribute
The colormap attribute specifies which colormap best reflects the true colors of the InputOutput window. The colormap must have the same visual type as the window, or a BadMatch error results. X servers capable of supporting multiple hardware colormaps can use this information, and window managers can use it for calls to XInstallColormap. You can set the colormap attribute to a colormap or to CopyFromParent (default).

If you set the colormap to CopyFromParent, the parent window’s colormap is copied and used by its child. However, the child window must have the same visual type as the parent, or a BadMatch error results. The parent window must not have a colormap of None, or a BadMatch error results. The colormap is copied by sharing the colormap object between the child and parent, not by making a complete copy of the colormap contents. Subsequent changes to the parent window’s colormap attribute do not affect the child window.

3.2.10. Cursor Attribute
The cursor attribute specifies which cursor is to be used when the pointer is in the InputOutput or InputOnly window. You can set the cursor to a cursor or None (default).

If you set the cursor to None, the parent’s cursor is used when the pointer is in the InputOutput or InputOnly window, and any change in the parent’s cursor will cause an immediate change in the displayed cursor. By calling XFreeCursor, the cursor can be freed immediately as long as no further explicit reference to it is made.

3.3. Creating Windows
Xlib provides basic ways for creating windows, and toolkits often supply higher-level functions specifically for creating and placing top-level windows, which are discussed in the appropriate toolkit documentation. If you do not use a toolkit, however, you must provide some standard information or hints for the window manager by using the Xlib inter-client communication functions (see chapter 14).

If you use Xlib to create your own top-level windows (direct children of the root window), you must observe the following rules so that all applications interact reasonably across the different styles of window management:

• You must never fight with the window manager for the size or placement of your top-level window.
• You must be able to deal with whatever size window you get, even if this means that your application just prints a message like “Please make me bigger” in its window.
• You should only attempt to resize or move top-level windows in direct response to a user request. If a request to change the size of a top-level window fails, you must be prepared to live with what you get. You are free to resize or move the children of top-level windows as necessary. (Toolkits often have facilities for automatic relayout.)
• If you do not use a toolkit that automatically sets standard window properties, you should set these properties for top-level windows before mapping them.

For further information, see chapter 14 and the Inter-Client Communication Conventions Manual.

**XCreateWindow** is the more general function that allows you to set specific window attributes when you create a window. **XCreateSimpleWindow** creates a window that inherits its attributes from its parent window.

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

To create an unmapped window and set its window attributes, use **XCreateWindow**.

**Window XCreateWindow**

```c
Display *display;
Window parent;
int x, y;
unsigned int width, height;
unsigned int border_width;
int depth;
unsigned int class;
Visual *visual;
unsigned long valuemask;
XSetWindowAttributes *attributes;
```

- **display** Specifies the connection to the X server.
- **parent** Specifies the parent window.
- **x y** Specify the x and y coordinates, which are the top-left outside corner of the created window’s borders and are relative to the inside of the parent window’s borders.
- **width height** Specify the width and height, which are the created window’s inside dimensions and do not include the created window’s borders. The dimensions must be nonzero, or a BadValue error results.
- **border_width** Specifies the width of the created window’s border in pixels.
- **depth** Specifies the window’s depth. A depth of CopyFromParent means the depth is taken from the parent.
- **class** Specifies the created window’s class. You can pass InputOutput, InputOnly, or CopyFromParent. A class of CopyFromParent means the class is taken from the parent.
- **visual** Specifies the visual type. A visual of CopyFromParent means the visual type is taken from the parent.
- **valuemask** Specifies which window attributes are defined in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero, the attributes are ignored and are not referenced.
- **attributes** Specifies the structure from which the values (as specified by the value mask) are to be taken. The value mask should have the appropriate bits set to indicate which attributes have been set in the structure.
The XCreateWindow function creates an unmapped subwindow for a specified parent window, returns the window ID of the created window, and causes the X server to generate a CreateNotify event. The created window is placed on top in the stacking order with respect to siblings.

The coordinate system has the X axis horizontal and the Y axis vertical, with the origin \([0, 0]\) at the upper left. Coordinates are integral, in terms of pixels, and coincide with pixel centers. Each window and pixmap has its own coordinate system. For a window, the origin is inside the border at the inside upper left.

The border_width for an InputOnly window must be zero, or a BadMatch error results. For class InputOutput, the visual type and depth must be a combination supported for the screen, or a BadMatch error results. The depth need not be the same as the parent, but the parent must not be a window of class InputOnly, or a BadMatch error results. For an InputOnly window, the depth must be zero, and the visual must be one supported by the screen. If either condition is not met, a BadMatch error results. The parent window, however, may have any depth and class. If you specify any invalid window attribute for a window, a BadMatch error results.

The created window is not yet displayed (mapped) on the user’s display. To display the window, call XMapWindow. The new window initially uses the same cursor as its parent. A new cursor can be defined for the new window by calling XDefineCursor. The window will not be visible on the screen unless it and all of its ancestors are mapped and it is not obscured by any of its ancestors.

XCreateWindow can generate BadAlloc, BadColor, BadCursor, BadMatch, BadPixmap, BadValue, and BadWindow errors.

To create an unmapped InputOutput subwindow of a given parent window, use XCreateSimpleWindow.

```
Window XCreateSimpleWindow(display, parent, x, y, width, height, border_width, border, background)
```

- **display** Specifies the connection to the X server.
- **parent** Specifies the parent window.
- **x** Specify the x and y coordinates, which are the top-left outside corner of the new window’s borders and are relative to the inside of the parent window’s borders.
- **y**
- **width** Specify the width and height, which are the created window’s inside dimensions and do not include the created window’s borders. The dimensions must be nonzero, or a BadValue error results.
- **height**
- **border_width** Specifies the width of the created window’s border in pixels.
- **border** Specifies the border pixel value of the window.
- **background** Specifies the background pixel value of the window.
The `XCreateSimpleWindow` function creates an unmapped InputOutput subwindow for a specified parent window, returns the window ID of the created window, and causes the X server to generate a `CreateNotify` event. The created window is placed on top in the stacking order with respect to siblings. Any part of the window that extends outside its parent window is clipped. The border_width for an InputOnly window must be zero, or a BadMatch error results. `XCreateSimpleWindow` inherits its depth, class, and visual from its parent. All other window attributes, except background and border, have their default values.

`XCreateSimpleWindow` can generate BadAlloc, BadMatch, BadValue, and BadWindow errors.

### 3.4. Destroying Windows

Xlib provides functions that you can use to destroy a window or destroy all subwindows of a window.

To destroy a window and all of its subwindows, use `XDestroyWindow`.

```c
XDestroyWindow(display, w)

Display *display;
Window w;
```

- `display` Specifies the connection to the X server.
- `w` Specifies the window.

The `XDestroyWindow` function destroys the specified window as well as all of its subwindows and causes the X server to generate a `DestroyNotify` event for each window. The window should never be referenced again. If the window specified by the `w` argument is mapped, it is unmapped automatically. The ordering of the `DestroyNotify` events is such that for any given window being destroyed, `DestroyNotify` is generated on any inferiors of the window before being generated on the window itself. The ordering among siblings and across subhierarchies is not otherwise constrained. If the window you specified is a root window, no windows are destroyed. Destroying a mapped window will generate Expose events on other windows that were obscured by the window being destroyed.

`XDestroyWindow` can generate a `BadWindow` error.

To destroy all subwindows of a specified window, use `XDestroySubwindows`.

```c
XDestroySubwindows(display, w)

Display *display;
Window w;
```

- `display` Specifies the connection to the X server.
- `w` Specifies the window.

The `XDestroySubwindows` function destroys all inferior windows of the specified window, in bottom-to-top stacking order. It causes the X server to generate a `DestroyNotify` event for each window. If any mapped subwindows were actually destroyed, `XDestroySubwindows` causes the X server to generate Expose events on the specified window. This is much more efficient than deleting many windows one at a time because much of the work need be performed only once for all of the windows, rather than for each window. The subwindows should never be referenced again.

`XDestroySubwindows` can generate a `BadWindow` error.

### 3.5. Mapping Windows

A window is considered mapped if an `XMapWindow` call has been made on it. It may not be visible on the screen for one of the following reasons:

---

32
• It is obscured by another opaque window.
• One of its ancestors is not mapped.
• It is entirely clipped by an ancestor.

Expose events are generated for the window when part or all of it becomes visible on the screen. A client receives the Expose events only if it has asked for them. Windows retain their position in the stacking order when they are unmapped.

A window manager may want to control the placement of subwindows. If SubstructureRedirectMask has been selected by a window manager on a parent window (usually a root window), a map request initiated by other clients on a child window is not performed, and the window manager is sent a MapRequest event. However, if the override-redirect flag on the child had been set to True (usually only on pop-up menus), the map request is performed.

A tiling window manager might decide to reposition and resize other clients’ windows and then decide to map the window to its final location. A window manager that wants to provide decoration might reparent the child into a frame first. For further information, see section 3.2.8 and section 10.10. Only a single client at a time can select for SubstructureRedirectMask.

Similarly, a single client can select for ResizeRedirectMask on a parent window. Then, any attempt to resize the window by another client is suppressed, and the client receives a ResizeRequest event.

To map a given window, use XMapWindow.

XMapWindow(display, w)

    Display *display;
    Window w;

display Specifies the connection to the X server.

w Specifies the window.

The XMapWindow function maps the window and all of its subwindows that have had map requests. Mapping a window that has an unmapped ancestor does not display the window but marks it as eligible for display when the ancestor becomes mapped. Such a window is called unviewable. When all its ancestors are mapped, the window becomes viewable and will be visible on the screen if it is not obscured by another window. This function has no effect if the window is already mapped.

If the override-redirect of the window is False and if some other client has selected SubstructureRedirectMask on the parent window, then the X server generates a MapRequest event, and the XMapWindow function does not map the window. Otherwise, the window is mapped, and the X server generates a MapNotify event.

If the window becomes viewable and no earlier contents for it are remembered, the X server tiles the window with its background. If the window’s background is undefined, the existing screen contents are not altered, and the X server generates zero or more Expose events. If backing-store was maintained while the window was unmapped, no Expose events are generated. If backing-store will now be maintained, a full-window exposure is always generated. Otherwise, only visible regions may be reported. Similar tiling and exposure take place for any newly viewable inferiors.

If the window is an InputOutput window, XMapWindow generates Expose events on each InputOutput window that it causes to be displayed. If the client maps and paints the window and if the client begins processing events, the window is painted twice. To avoid this, first ask for Expose events and then map the window, so the client processes input events as usual. The event list will include Expose for each window that has appeared on the screen. The client’s normal response to an Expose event should be to repaint the window. This method usually leads to simpler programs and to proper interaction with window managers.
Xlib - C Library

XMapWindow can generate a BadWindow error.

To map and raise a window, use XMapRaised.

\[ \text{XMapRaised}(\text{display}, w) \]

- \text{Display \*display;}
- \text{Window w;}

\text{display} \quad \text{Specifies the connection to the X server.}
\text{w} \quad \text{Specifies the window.}

The \text{XMapRaised} function essentially is similar to \text{XMapWindow} in that it maps the window and all of its subwindows that have had map requests. However, it also raises the specified window to the top of the stack. For additional information, see \text{XMapWindow}.

\text{XMapRaised} can generate multiple \text{BadWindow} errors.

To map all subwindows for a specified window, use \text{XMapSubwindows}.

\[ \text{XMapSubwindows}(\text{display}, w) \]

- \text{Display \*display;}
- \text{Window w;}

\text{display} \quad \text{Specifies the connection to the X server.}
\text{w} \quad \text{Specifies the window.}

The \text{XMapSubwindows} function maps all subwindows for a specified window in top-to-bottom stacking order. The X server generates Expose events on each newly displayed window. This may be much more efficient than mapping many windows one at a time because the server needs to perform much of the work only once, for all of the windows, rather than for each window.

\text{XMapSubwindows} can generate a \text{BadWindow} error.

3.6. Unmapping Windows

Xlib provides functions that you can use to unmap a window or all subwindows.

To unmap a window, use \text{XUnmapWindow}.

\[ \text{XUnmapWindow}(\text{display}, w) \]

- \text{Display \*display;}
- \text{Window w;}

\text{display} \quad \text{Specifies the connection to the X server.}
\text{w} \quad \text{Specifies the window.}

The \text{XUnmapWindow} function unmaps the specified window and causes the X server to generate an \text{UnmapNotify} event. If the specified window is already unmapped, \text{XUnmapWindow} has no effect. Normal exposure processing on formerly obscured windows is performed. Any child window will no longer be visible until another map call is made on the parent. In other words, the subwindows are still mapped but are not visible until the parent is mapped. Unmapping a window will generate Expose events on windows that were formerly obscured by it.

\text{XUnmapWindow} can generate a \text{BadWindow} error.

To unmap all subwindows for a specified window, use \text{XUnmapSubwindows}.
Xlib - C Library

XUnmapSubwindows(display, w)

Display *display;
Window w;

display Specifies the connection to the X server.
w Specifies the window.

The XUnmapSubwindows function unmaps all subwindows for the specified window in bottom-to-top stacking order. It causes the X server to generate an UnmapNotify event on each subwindow and Expose events on formerly obscured windows. Using this function is much more efficient than unmapping multiple windows one at a time because the server needs to perform much of the work only once, for all of the windows, rather than for each window. XUnmapSubwindows can generate a BadWindow error.

3.7. Configuring Windows

Xlib provides functions that you can use to move a window, resize a window, move and resize a window, or change a window's border width. To change one of these parameters, set the appropriate member of the XWindowChanges structure and OR in the corresponding value mask in subsequent calls to XConfigureWindow. The symbols for the value mask bits and the XWindowChanges structure are:

/* Configure window value mask bits */

#define CWX (1<<0)
#define CWY (1<<1)
#define CWWidth (1<<2)
#define CWHeight (1<<3)
#define CBWidth (1<<4)
#define CWBorderWidth (1<<5)
#define CWStackMode (1<<6)

/* Values */

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

The x and y members are used to set the window's x and y coordinates, which are relative to the parent's origin and indicate the position of the upper-left outer corner of the window. The width and height members are used to set the inside size of the window, not including the border, and must be nonzero, or a BadValue error results. Attempts to configure a root window have no effect.

The border_width member is used to set the width of the border in pixels. Note that setting 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. If you attempt to set the border-width attribute of an InputOnly window nonzero, a BadMatch error results.

The sibling member is used to set the sibling window for stacking operations. The stack_mode member is used to set how the window is to be restacked and can be set to Above, Below, Toplf, Bottomlf, or Opposite.

If the override-redirect flag of the window is False and if some other client has selected SubstructureRedirectMask on the parent, the X server generates a ConfigureRequest event, and
no further processing is performed. Otherwise, if some other client has selected `ResizeRedirectMask` on the window and the inside width or height of the window is being changed, a `ResizeRequest` event is generated, and the current inside width and height are used instead. Note that the override-transparent flag of the window has no effect on `ResizeRedirectMask` and that `SubstructureRedirectMask` on the parent has precedence over `ResizeRedirectMask` on the window.

When the geometry of the window is changed as specified, the window is restacked among siblings, and a `ConfigureNotify` event is generated if the state of the window actually changes. `GravityNotify` events are generated after `ConfigureNotify` events. If the inside width or height of the window has actually changed, children of the window are affected as specified.

If a window’s size actually changes, the window’s subwindows move according to their window gravity. Depending on the window’s bit gravity, the contents of the window also may be moved (see section 3.2.3).

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

The restack check (specifically, the computation for `BottomIf`, `TopIf`, and `Opposite`) is performed with respect to the window’s final size and position (as controlled by the other arguments of the request), not its initial position. If a sibling is specified without a stack_mode, a BadMatch error results.

If a sibling and a stack_mode are specified, the window is restacked as follows:

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

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

- **Above**
  - The window is placed at the top of the stack.
  
- **Below**
  - The window is placed at the bottom of the stack.
  
- **TopIf**
  - If any sibling occludes the window, the window is placed at the top of the stack.
  
- **BottomIf**
  - If the window occludes any sibling, the window is placed at the bottom of the stack.
  
- **Opposite**
  - If any sibling occludes the window, the window is placed at the top of the stack. If the window occludes any sibling, the window is placed at the bottom of the stack.

Attempts to configure a root window have no effect.

To configure a window’s size, location, stacking, or border, use `XConfigureWindow`.
XConfigureWindow(display, w, value_mask, values)
  Display *display;
  Window w;
  unsigned int value_mask;
  XWindowChanges *values;

  display Specifies the connection to the X server.
  w Specifies the window to be reconfigured.
  value_mask Specifies which values are to be set using information in the values structure.
    This mask is the bitwise inclusive OR of the valid configure window values
    bits.
  values Specifies the XWindowChanges structure.

The XConfigureWindow function uses the values specified in the XWindowChanges structure to reconfigure a window's size, position, border, and stacking order. Values not specified are taken from the existing geometry of the window.

If a sibling is specified without a stack_mode or if the window is not actually a sibling, a BadMatch error results. Note that the computations for BottomIf, TopIf, and Opposite are performed with respect to the window's final geometry (as controlled by the other arguments passed to XConfigureWindow), not its initial geometry. Any backing store contents of the window, its inferiors, and other newly visible windows are either discarded or changed to reflect the current screen contents (depending on the implementation).

XConfigureWindow can generate BadMatch, BadValue, and BadWindow errors.

To move a window without changing its size, use XMoveWindow.

XMoveWindow(display, w, x, y)
  Display *display;
  Window w;
  int x, y;

  display Specifies the connection to the X server.
  w Specifies the window to be moved.
  x y Specify the x and y coordinates, which define the new location of the top-left
    pixel of the window's border or the window itself if it has no border.

The XMoveWindow function moves the specified window to the specified x and y coordinates, but it does not change the window's size, raise the window, or change the mapping state of the window. Moving a mapped window may or may not lose the window's contents depending on if the window is obscured by nonchildren and if no backing store exists. If the contents of the window are lost, the X server generates Expose events. Moving a mapped window generates Expose events on any formerly obscured windows.

If the override-redirect flag of the window is False and some other client has selected SubstructureRedirectMask on the parent, the X server generates a ConfigureRequest event, and no further processing is performed. Otherwise, the window is moved.

XMoveWindow can generate a BadWindow error.

To change a window's size without changing the upper-left coordinate, use XResizeWindow.

XResizeWindow(display, w, width, height)
  Display *display;
  Window w;
  unsigned int width, height;
**Xlib – C Library**

**XI1, Release 5**

*display*  
Specifies the connection to the X server.

*w*  
Specifies the window.

*height*  
Specifies the width and height, which are the interior dimensions of the window after the call completes.

The **XResizeWindow** function changes the inside dimensions of the specified window, not including its borders. This function does not change the window’s upper-left coordinate or the origin and does not restack the window. Changing the size of a mapped window may lose its contents and generate **Expose** events. If a mapped window is made smaller, changing its size generates **Expose** events on windows that the mapped window formerly obscured.

If the override-redirect flag of the window is **False** and some other client has selected **SubstructureRedirectMask** on the parent, the X server generates a **ConfigureRequest** event, and no further processing is performed. If either width or height is zero, a **BadValue** error results.

**XResizeWindow** can generate **BadValue** and **BadWindow** errors.

To change the size and location of a window, use **XMoveResizeWindow**.

**XMoveResizeWindow**(*display*, w, x, y, width, height)

*display*  
Specifies the connection to the X server.

*w*  
Specifies the window to be reconfigured.

*x*  
Specifies the x coordinate, which defines the new position of the window relative to its parent.

*y*  
Specifies the y coordinate, which defines the new position of the window relative to its parent.

*width*  
Specifies the width and height, which define the interior size of the window.

The **XMoveResizeWindow** function changes the size and location of the specified window without raising it. Moving and resizing a mapped window may generate an **Expose** event on the window. Depending on the new size and location parameters, moving and resizing a window may generate **Expose** events on windows that the window formerly obscured.

If the override-redirect flag of the window is **False** and some other client has selected **SubstructureRedirectMask** on the parent, the X server generates a **ConfigureRequest** event, and no further processing is performed. Otherwise, the window size and location are changed.

**XMoveResizeWindow** can generate **BadValue** and **BadWindow** errors.

To change the border width of a given window, use **XSetWindowBorderWidth**.

**XSetWindowBorderWidth**(*display*, w, width)

*display*  
Specifies the connection to the X server.

*w*  
Specifies the window.

*width*  
Specifies the width of the window border.

The **XSetWindowBorderWidth** function sets the specified window’s border width to the specified width.
XSetWindowBorderWidth can generate a BadWindow error.

3.8. Changing Window Stacking Order
Xlib provides functions that you can use to raise, lower, circulate, or restack windows.

To raise a window so that no sibling window obscures it, use XRaiseWindow.

XRaiseWindow(display, w)
  Display *display;
  Window w;

  display Specifies the connection to the X server.
  w Specifies the window.

The XRaiseWindow function raises the specified window to the top of the stack so that no sibling window obscures it. If the windows are regarded as overlapping sheets of paper stacked on a desk, then raising a window is analogous to moving the sheet to the top of the stack but leaving its x and y location on the desk constant. Raising a mapped window may generate Expose events for the window and any mapped subwindows that were formerly obscured.

If the override-redirect attribute of the window is False and some other client has selected SubstructureRedirectMask on the parent, the X server generates a ConfigureRequest event, and no processing is performed. Otherwise, the window is raised.

XRaiseWindow can generate a BadWindow error.

To lower a window so that it does not obscure any sibling windows, use XLowerWindow.

XLowerWindow(display, w)
  Display *display;
  Window w;

  display Specifies the connection to the X server.
  w Specifies the window.

The XLowerWindow function lowers the specified window to the bottom of the stack so that it does not obscure any sibling windows. If the windows are regarded as overlapping sheets of paper stacked on a desk, then lowering a window is analogous to moving the sheet to the bottom of the stack but leaving its x and y location on the desk constant. Lowering a mapped window will generate Expose events on any windows it formerly obscured.

If the override-redirect attribute of the window is False and some other client has selected SubstructureRedirectMask on the parent, the X server generates a ConfigureRequest event, and no processing is performed. Otherwise, the window is lowered to the bottom of the stack.

XLowerWindow can generate a BadWindow error.

To circulate a subwindow up or down, use XCirculateSubwindows.

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

  display Specifies the connection to the X server.
  w Specifies the window.
  direction Specifies the direction (up or down) that you want to circulate the window.
  You can pass RaiseLowest or LowerHighest.
The `X CirculateSubwindows` function circulates children of the specified window in the specified direction. If you specify `RaiseLowest`, `X CirculateSubwindows` raises the lowest mapped child (if any) that is occluded by another child to the top of the stack. If you specify `LowerHighest`, `X CirculateSubwindows` lowers the highest mapped child (if any) that occludes another child to the bottom of the stack. Exposure processing is then performed on formerly obscured windows. If some other client has selected `SubstructureRedirectMask` on the window, the X server generates a `CirculateRequest` event, and no further processing is performed. If a child is actually restacked, the X server generates a `CirculateNotify` event. `X CirculateSubwindows` can generate `BadValue` and `BadWindow` errors.

To raise the lowest mapped child of a window that is partially or completely occluded by another child, use `X CirculateSubwindowsUp`.

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

display Specifies the connection to the X server.
w Specifies the window.
```

The `X CirculateSubwindowsUp` function raises the lowest mapped child of the specified window that is partially or completely occluded by another child. Completely unobscured children are not affected. This is a convenience function equivalent to `X CirculateSubwindows` with `RaiseLowest` specified.

`X CirculateSubwindowsUp` can generate a `BadWindow` error.

To lower the highest mapped child of a window that partially or completely occludes another child, use `X CirculateSubwindowsDown`.

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

display Specifies the connection to the X server.
w Specifies the window.
```

The `X CirculateSubwindowsDown` function lowers the highest mapped child of the specified window that partially or completely occludes another child. Completely unobscured children are not affected. This is a convenience function equivalent to `X CirculateSubwindows` with `LowerHighest` specified.

`X CirculateSubwindowsDown` can generate a `BadWindow` error.

To restack a set of windows from top to bottom, use `X RestackWindows`.

```
xrestackWindows(display, windows, nwindows)
    Display *display;
    Window windows[];
    int nwindows;

display Specifies the connection to the X server.
windows Specifies an array containing the windows to be restacked.
nwindows Specifies the number of windows to be restacked.
```

The `X RestackWindows` function restacks the windows in the order specified, from top to bottom. The stacking order of the first window in the `windows` array is unaffected, but the other windows in the array are stacked underneath the first window, in the order of the array. The stacking order of the other windows is not affected. For each window in the window array
that is not a child of the specified window, a BadMatch error results.

If the override-redirect attribute of a window is False and some other client has selected SubstructureRedirectMask on the parent, the X server generates ConfigureRequest events for each window whose override-redirect flag is not set, and no further processing is performed. Otherwise, the windows will be restacked in top to bottom order. XRestackWindows can generate a BadWindow error.

3.9. Changing Window Attributes

Xlib provides functions that you can use to set window attributes. XChangeWindowAttributes is the more general function that allows you to set one or more window attributes provided by the XSetWindowAttributes structure. The other functions described in this section allow you to set one specific window attribute, such as a window’s background.

To change one or more attributes for a given window, use XChangeWindowAttributes.

XChangeWindowAttributes(display, w, valuemask, attributes)

display specifies the connection to the X server.
w specifies the window.
valuemask specifies which window attributes are defined in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero, the attributes are ignored and are not referenced. The values and restrictions are the same as for XCreateWindow.

attributes specifies the structure from which the values (as specified by the value mask) are to be taken. The value mask should have the appropriate bits set to indicate which attributes have been set in the structure (see section 3.2).

Depending on the valuemask, the XChangeWindowAttributes function uses the window attributes in the XSetWindowAttributes structure to change the specified window attributes. Changing the background does not cause the window contents to be changed. To repaint the window and its background, use XClearWindow. 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 docs 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. Changing the colormap of a window (that is, defining a new map, not changing the contents of the existing map) generates a ColormapNotify event. Changing the colormap of a visible window may have no immediate effect on the screen because the map may not be installed (see XInstallColormap). Changing the cursor of a root window to None restores the default cursor. Whenever possible, you are encouraged to share colormaps.

Multiple clients can select input on the same window. Their event masks are maintained separately. When an event is generated, it is reported to all interested clients. However, only one client at a time can select for SubstructureRedirectMask, ResizeRedirectMask, and ButtonPressMask. If a client attempts to select any of these event masks and some other client has already selected one, a BadAccess error results. There is only one do-not-propagate-mask for a window, not one per client.
XChangeWindowAttributes can generate BadAccess, BadColor, BadCursor, BadMatch, BadPixmap, BadValue, and BadWindow errors.

To set the background of a window to a given pixel, use XSetWindowBackground.

XSetWindowBackground(display, w, background_pixel)
   Display *display;
   Window w;
   unsigned long background_pixel;

   display Specifies the connection to the X server.
   w Specifies the window.
   background_pixel Specifies the pixel that is to be used for the background.

The XSetWindowBackground function sets the background of the window to the specified pixel value. Changing the background does not cause the window contents to be changed. XSetWindowBackground uses a pixmap of undefined size filled with the pixel value you passed. If you try to change the background of an InputOnly window, a BadMatch error results. XSetWindowBackground can generate BadMatch and BadWindow errors.

To set the background of a window to a given pixmap, use XSetWindowBackgroundPixmap.

XSetWindowBackgroundPixmap(display, w, backgroundPixmap)
   Display *display;
   Window w;
  Pixmap backgroundPixmap;

   display Specifies the connection to the X server.
   w Specifies the window.
   backgroundPixmap Specifies the background pixmap, ParentRelative, or None.

The XSetWindowBackgroundPixmap function sets the background pixmap of the window to the specified pixmap. The background pixmap can immediately be freed if no further explicit references to it are to be made. If ParentRelative is specified, the background pixmap of the window’s parent is used, or on the root window, the default background is restored. If you try to change the background of an InputOnly window, a BadMatch error results. If the background is set to None, the window has no defined background.

XSetWindowBackgroundPixmap can generate BadMatch, BadPixmap, and BadWindow errors.

Note

XSetWindowBackground and XSetWindowBackgroundPixmap do not change the current contents of the window.

To change and repaint a window’s border to a given pixel, use XSetWindowBorder.

XSetWindowBorder(display, w, border_pixel)
   Display *display;
   Window w;
   unsigned long border_pixel;
display Specifies the connection to the X server.

w Specifies the window.

border_pixel Specifies the entry in the colormap.

The XSetWindowBorder function sets the border of the window to the pixel value you specify. If you attempt to perform this on an InputOnly window, a BadMatch error results. XSetWindowBorder can generate BadMatch and BadWindow errors.

To change and repaint the border tile of a given window, use XSetWindowBorderPixmap.

XSetWindowBorderPixmap(display, w, border_pixmap)

Display *display;
Window w;
Pixmap border_pixmap;

display Specifies the connection to the X server.

w Specifies the window.

border_pixmap Specifies the border pixmap or CopyFromParent.

The XSetWindowBorderPixmap function sets the border pixmap of the window to the pixmap you specify. The border pixmap can be freed immediately if no further explicit references to it are to be made. If you specify CopyFromParent, a copy of the parent window’s border pixmap is used. If you attempt to perform this on an InputOnly window, a BadMatch error results.

XSetWindowBorderPixmap can generate BadMatch, BadPixmap, and BadWindow errors.

To set the colormap of a given window, use XSetWindowColormap.

XSetWindowColormap(display, w, colormap)

Display *display;
Window w;
Colormap colormap;

display Specifies the connection to the X server.

w Specifies the window.

colormap Specifies the colormap.

The XSetWindowColormap function sets the specified colormap of the specified window. The colormap must have the same visual type as the window, or a BadMatch error results. XSetWindowColormap can generate BadColor, BadMatch, and BadWindow errors.

To define which cursor will be used in a window, use XDefineCursor.

XDefineCursor(display, w, cursor)

Display *display;
Window w;
Cursor cursor;

display Specifies the connection to the X server.

w Specifies the window.

cursor Specifies the cursor that is to be displayed or None.

If a cursor is set, it will be used when the pointer is in the window. If the cursor is None, it is equivalent to XUndefineCursor.

XDefineCursor can generate BadCursor and BadWindow errors.
To undefine the cursor in a given window, use XUndefineCursor.

XUndefineCursor(display, w)
    Display *display;
    Window w;

display    Specifies the connection to the X server.
w          Specifies the window.

The XUndefineCursor function undoes the effect of a previous XDefineCursor for this window. When the pointer is in the window, the parent’s cursor will now be used. On the root window, the default cursor is restored.

XUndefineCursor can generate a BadWindow error.
Chapter 4

Window Information Functions

After you connect the display to the X server and create a window, you can use the Xlib window information functions to:

- Obtain information about a window
- Translate screen coordinates
- Manipulate property lists
- Obtain and change window properties
- Manipulate selections

4.1. Obtaining Window Information

Xlib provides functions that you can use to obtain information about the window tree, the window’s current attributes, the window’s current geometry, or the current pointer coordinates. Because they are most frequently used by window managers, these functions all return a status to indicate whether the window still exists.

To obtain the parent, a list of children, and number of children for a given window, use XQueryTree.

Status XQueryTree(display, w, root_return, parent_return, children_return, nchildren_return)

- Display *display;
- Window w;
- Window *root_return;
- Window *parent_return;
- Window **children_return;
- unsigned int *nchildren_return;

*display* Specifies the connection to the X server.

*w* Specifies the window whose list of children, root, parent, and number of children you want to obtain.

*root_return* Returns the root window.

*parent_return* Returns the parent window.

*children_return* Returns the list of children.

*nchildren_return* Returns the number of children.

The XQueryTree function returns the root ID, the parent window ID, a pointer to the list of children windows, and the number of children in the list for the specified window. The children are listed in current stacking order, from bottommost (first) to topmost (last). XQueryTree returns zero if it fails and nonzero if it succeeds. To free this list when it is no longer needed, use XFree.

XQueryTree can generate a BadWindow error.

To obtain the current attributes of a given window, use XGetWindowAttributes.
Status XGetWindowAttributes(display, w, window_attributes_return)

Display *display;
Window w;
XWindowAttributes *window_attributes_return;

display Specifies the connection to the X server.
w Specifies the window whose current attributes you want to obtain.

window_attributes_return Returns the specified window’s attributes in the XWindowAttributes structure.

The XGetWindowAttributes function returns the current attributes for the specified window to an XWindowAttributes structure.

typedef struct {
    int x, y; /* location of window */
    int width, height; /* width and height of window */
    int border_width; /* border width of window */
    int depth; /* depth of window */
    Visual *visual; /* the associated visual structure */
    Window root; /* root of screen containing window */
    int class; /* InputOutput, InputOnly */
    int bit_gravity; /* one of the bit gravity values */
    int win_gravity; /* one of the window gravity values */
    int backing_store; /* NotUseful, WhenMapped, Always */
    unsigned long backing_planes; /* planes to be preserved if possible */
    unsigned long backing_pixel; /* value to be used when restoring planes */
    Bool save_under; /* boolean, should bits under be saved? */
    Colormap colormap; /* color map to be associated with window */
    Bool map_installed; /* boolean, is color map currently installed */
    int map_state; /* IsUnmapped, IsUnviewable, IsViewable */
    long all_event_masks; /* set of events all people have interest in */
    long your_event_mask; /* my event mask */
    long do_not_propagate_mask; /* set of events that should not propagate */
    Bool override_redirect; /* boolean value for override-redirect */
    Screen *screen; /* back pointer to correct screen */
} XWindowAttributes;

The x and y members are set to the upper-left outer corner relative to the parent window’s origin. The width and height members are set to the inside size of the window, not including the border. The border_width member is set to the window’s border width in pixels. The depth member is set to the depth of the window (that is, bits per pixel for the object). The visual member is a pointer to the screen’s associated Visual structure. The root member is set to the root window of the screen containing the window. The class member is set to the window’s class and can be either InputOutput or InputOnly.

The bit_gravity member is set to the window’s bit gravity and can be one of the following:

ForgetGravity EastGravity
NorthWestGravity SouthWestGravity
NorthGravity SouthGravity
NorthEastGravity SouthEastGravity
WestGravity StaticGravity
CenterGravity

The win_gravity member is set to the window’s window gravity and can be one of the following:
Xlib – C Library

UnmapGravity
NorthWestGravity
NorthGravity
NorthEastGravity
WestGravity
CenterGravity

For additional information on gravity, see section 3.3.

The backing_store member is set to indicate how the X server should maintain the contents of a window and can be WhenMapped, Always, or NotUseful. The backing_planes member is set to indicate (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 member is set to indicate what values to use for planes not set in backing_planes.

The save_under member is set to True or False. The colormap member is set to the colormap for the specified window and can be a colormap ID or None. The map_installed member is set to indicate whether the colormap is currently installed and can be True or False. The map_state member is set to indicate the state of the window and can be IsUnmapped, IsUnviewable, or IsViewable. IsUnviewable is used if the window is mapped but some ancestor is unmapped.

The all_event_masks member is set to the bitwise inclusive OR of all event masks selected on the window by all clients. The your_event_mask member is set to the bitwise inclusive OR of all event masks selected by the querying client. The do_not_propagate_mask member is set to the bitwise inclusive OR of the set of events that should not propagate.

The override_redirect member is set to indicate whether this window overrides structure control facilities and can be True or False. Window manager clients should ignore the window if this member is True.

The screen member is set to a screen pointer that gives you a back pointer to the correct screen. This makes it easier to obtain the screen information without having to loop over the root window fields to see which field matches.

XGetWindowAttributes can generate BadDrawable and BadWindow errors.

To obtain the current geometry of a given drawable, use XGetGeometry.

Status XGetGeometry(display, d, root_return, x_return, y_return, width_return, height_return, border_width_return, depth_return)

Display *display;
Drawable d;
Window *root_return;
int *x_return, *y_return;
unsigned int *width_return, *height_return;
unsigned int *border_width_return;
unsigned int *depth_return;

display Specifies the connection to the X server.
d Specifies the drawable, which can be a window or a pixmap.
root_return Returns the root window.
x_return Return the x and y coordinates that define the location of the drawable. For a window, these coordinates specify the upper-left outer corner relative to its parent’s origin. For pixmaps, these coordinates are always zero.
y_return
width_return
height_return Return the drawable’s dimensions (width and height). For a window, these dimensions specify the inside size, not including the border.
border_width_return
Returns the border width in pixels. If the drawable is a pixmap, it returns zero.

depth_return
Returns the depth of the drawable (bits per pixel for the object).
The XGetGeometry function returns the root window and the current geometry of the draw-
able. The geometry of the drawable includes the x and y coordinates, width and height, border
width, and depth. These are described in the argument list. It is legal to pass to this function
a window whose class is InputOnly.
XGetGeometry can generate a BadDrawable error.

4.2. Translating Screen Coordinates
Applications sometimes need to perform a coordinate transformation from the coordinate space
of one window to another window or need to determine which window the pointing device is
in. XTranslateCoordinates and XQueryPointer fulfill these needs (and avoids any race con-
ditions) by asking the X server to perform these operations.

To translate a coordinate in one window to the coordinate space of another window, use
XTranslateCoordinates.

Bool XTranslateCoordinates(display, src_w, dest_w, src_x, src_y, dest_x_return,
                          dest_y_return, child_return)

  Display *display;
  Window src_w, dest_w;
  int src_x, src_y;
  int *dest_x_return, *dest_y_return;
  Window *child_return;

display Specifies the connection to the X server.
src_w Specifies the source window.
dest_w Specifies the destination window.
src_x Specify the x and y coordinates within the source window.
src_y dest_x_return Return the x and y coordinates within the destination window.
dest_y_return Returns the child if the coordinates are contained in a mapped child of the des-
tination window.

child_return If XTranslateCoordinates returns True, it takes the src_x and src_y coordinates relative to
the source window’s origin and returns these coordinates to dest_x_return and dest_y_return
relative to the destination window’s origin. If XTranslateCoordinates returns False, src_w
and dest_w are on different screens, and dest_x_return and dest_y_return are zero. If the coor-
dinates are contained in a mapped child of dest_w, that child is returned to child_return. Oth-
erwise, child_return is set to None.
XTranslateCoordinates can generate a BadWindow error.

To obtain the screen coordinates of the pointer, or to determine the pointer coordinates relative
to a specified window, use XQueryPointer.
**Bool XQueryPointer**

```
  display, w, root_return, child_return, root_x_return, root_y_return,
  win_x_return, win_y_return, mask_return)
```

This function returns the root window the pointer is logically on and the pointer coordinates relative to the root window's origin. If `XQueryPointer` returns `False`, the pointer is not on the same screen as the specified window, and `XQueryPointer` returns `None` to `child_return` and zero to `win_x_return` and `win_y_return`. If `XQueryPointer` returns `True`, the pointer coordinates returned to `win_x_return` and `win_y_return` are relative to the origin of the specified window. In this case, `XQueryPointer` returns the child that contains the pointer, if any, or else `None` to `child_return`.

`XQueryPointer` returns the current logical state of the keyboard buttons and the modifier keys in `mask_return`. It sets `mask_return` to the bitwise inclusive OR of one or more of the button or modifier key bitmasks to match the current state of the mouse buttons and the modifier keys.

Note that the logical state of a device (as seen through Xlib) may lag the physical state if device event processing is frozen (see section 12.1).

`XQueryPointer` can generate a `BadWindow` error.

### 4.3. Properties and Atoms

A property is a collection of named, typed data. The window system has a set of predefined properties (for example, the name of a window, size hints, and so on), and users can define any other arbitrary information and associate it with windows. Each property has a name, which is an ISO Latin-1 string. For each named property, a unique identifier (atom) is associated with it. A property also has a type, for example, string or integer. These types are also indicated using atoms, so arbitrary new types can be defined. Data of only one type may be associated with a single property name. Clients can store and retrieve properties associated with windows. For efficiency reasons, an atom is used rather than a character string. `XInternAtom` can be used to obtain the atom for property names.

A property is also stored in one of several possible formats. The X server can store the information as 8-bit quantities, 16-bit quantities, or 32-bit quantities. This permits the X server to
present the data in the byte order that the client expects.

Note

If you define further properties of complex type, you must encode and decode them yourself. These functions must be carefully written if they are to be portable.

For further information about how to write a library extension, see appendix C.

The type of a property is defined by an atom, which allows for arbitrary extension in this type scheme.

Certain property names are predefined in the server for commonly used functions. The atoms for these properties are defined in <X11/Xatom.h>. To avoid name clashes with user symbols, the #define name for each atom has the XA_ prefix. For definitions of these properties, see section 4.3. For an explanation of the functions that let you get and set much of the information stored in these predefined properties, see chapter 14.

The core protocol imposes no semantics on these property 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.

You can use properties to communicate other information between applications. The functions described in this section let you define new properties and get the unique atom IDs in your applications.

Although any particular atom can have some client interpretation within each of the name spaces, atoms occur in five distinct name spaces within the protocol:

- Selections
- Property names
- Property types
- Font properties
- Type of a ClientMessage event (none are built into the X server)

The built-in selection property names are:

- PRIMARY
- SECONDARY

The built-in property names are:

- CUT_BUFFER0
- CUT_BUFFER1
- CUT_BUFFER2
- CUT_BUFFER3
- CUT_BUFFER4
- CUT_BUFFER5
- CUT_BUFFER6
- CUT_BUFFER7
- RGB_DEFAULT_MAP
- RGB_RED_MAP
- RGB_GREEN_MAP
- RGB_BLUE_MAP
- RGB_GRAY_MAP
- RGB_BEST_MAP
- RGB_DEFAULT_MAP

The built-in property types are:
The built-in font property names are:

- MIN_SPACE
- NORM_SPACE
- MAX_SPACE
- END_SPACE
- SUPERSCRIPT_X
- SUPERSCRIPT_Y
- SUBSCRIPT_X
- SUBSCRIPT_Y
- UNDERLINE_POSITION
- UNDERLINE_THICKNESS
- FONT_NAME
- FULL_NAME

For further information about font properties, see section 8.5.

To return an atom for a given name, use XInternAtom.

Atom XInternAtom(display, atomname, only_if_exists)
  Display *display;
  char *atomname;
  Bool only_if_exists;

  display Specifies the connection to the X server.
  atomname Specifies the name associated with the atom you want returned.
  only_if_exists Specifies a Boolean value that indicates whether XInternAtom creates the atom.

The XInternAtom function returns the atom identifier associated with the specified atomname string. If only_if_exists is False, the atom is created if it does not exist. Therefore, XInternAtom can return None. If the atom name is not in the Host Portable Character Encoding the result is implementation dependent. Case matters; the strings thing, Thing, and thinG all designate different atoms. The atom will remain defined even after the client's connection closes. It will become undefined only when the last connection to the X server closes.

XInternAtom can generate BadAlloc and BadValue errors.

To return a name for a given atom identifier, use XGetAtomName.

char *XGetAtomName(display, atom)
  Display *display;
  Atom atom;
The **XGetAtomName** function returns the name associated with the specified atom. If the data returned by the server is in the Latin Portable Character Encoding, then the returned string is in the Host Portable Character Encoding. Otherwise, the result is implementation dependent. To free the resulting string, call **XFree**.

**XGetAtomName** can generate a **BadAtom** error.

### 4.4. Obtaining and Changing Window Properties

You can attach a property list to every window. Each property has a name, a type, and a value (see section 4.3). The value is an array of 8-bit, 16-bit, or 32-bit quantities, whose interpretation is left to the clients.

Xlib provides functions that you can use to obtain, change, update, or interchange window properties. In addition, Xlib provides other utility functions for inter-client communication (see chapter 14).

To obtain the type, format, and value of a property of a given window, use **XGetWindowProperty**.

```c
int XGetWindowProperty(display, w, property, long_offset, long_length, delete, req_type,
                         actual_type_return, actual_format_return, nitems_return, bytes_after_return,
                         prop_return)

Display *display;
Window w;
Atom property;
long long_offset, long_length;
Bool delete;
Atom req_type;
Atom *actual_type_return;
int *actual_format_return;
unsigned long *nitems_return;
unsigned long *bytes_after_return;
unsigned char **prop_return;
```

- **display**: Specifies the connection to the X server.
- **w**: Specifies the window whose property you want to obtain.
- **property**: Specifies the property name.
- **long_offset**: Specifies the offset in the specified property (in 32-bit quantities) where the data is to be retrieved.
- **long_length**: Specifies the length in 32-bit multiples of the data to be retrieved.
- **delete**: Specifies a Boolean value that determines whether the property is deleted.
- **req_type**: Specifies the atom identifier associated with the property type or **AnyPropertyType**.
- **actual_type_return**: Returns the atom identifier that defines the actual type of the property.
- **actual_format_return**: Returns the actual format of the property.
- **nitems_return**: Returns the actual number of 8-bit, 16-bit, or 32-bit items stored in the prop_return data.
- **bytes_after_return**: Returns the number of bytes remaining to be read in the property if a partial
read was performed.

**prop_return** Returns the data in the specified format.

The `XGetWindowProperty` function returns the actual type of the property; the actual format of the property; the number of 8-bit, 16-bit, or 32-bit items transferred; the number of bytes remaining to be read in the property; and a pointer to the data actually returned. `XGetWindowProperty` sets the return arguments as follows:

- If the specified property does not exist for the specified window, `XGetWindowProperty` returns `None` to `actual_type_return` and the value zero to `actual_format_return` and `bytes_after_return`. The `nitems_return` argument is empty. In this case, the `delete` argument is ignored.

- If the specified property exists but its type does not match the specified type, `XGetWindowProperty` returns the actual property type to `actual_type_return`, the actual property format (never zero) to `actual_format_return`, and the property length in bytes (even if the `actual_format_return` is 16 or 32) to `bytes_after_return`. It also ignores the `delete` argument. The `nitems_return` argument is empty.

- If the specified property exists and either you assign `AnyPropertyType` to the `req_type` argument or the specified type matches the actual property type, `XGetWindowProperty` returns the actual property type to `actual_type_return` and the actual property format (never zero) to `actual_format_return`. It also returns a value to `bytes_after_return` and `nitems_return`, by defining the following values:

  \[
  \begin{align*}
  N &= \text{actual length of the stored property in bytes} \quad \text{(even if the format is 16 or 32)} \\
  I &= 4 \times \text{long_offset} \\
  T &= N - I \\
  L &= \text{MINIMUM}(T, 4 \times \text{long_length}) \\
  A &= N - (I + L)
  \end{align*}
  \]

  The returned value starts at byte index I in the property (indexing from zero), and its length in bytes is L. If the value for `long_offset` causes L to be negative, a `BadValue` error results. The value of `bytes_after_return` is A, giving the number of trailing unread bytes in the stored property.

`XGetWindowProperty` always allocates one extra byte in `prop_return` (even if the property is zero length) and sets it to ASCII null so that simple properties consisting of characters do not have to be copied into yet another string before use. If `delete` is `True` and `bytes_after_return` is zero, `XGetWindowProperty` deletes the property from the window and generates a `PropertyNotify` event on the window.

The function returns `Success` if it executes successfully. To free the resulting data, use `XFree`.

`XGetWindowProperty` can generate `BadAtom`, `BadValue`, and `BadWindow` errors.

To obtain a given window’s property list, use `XListProperties`.

\[
\text{Atom} * \text{XListProperties}(\text{display, w, num\_prop\_return})
\]

\[
\begin{align*}
\text{Display} * & \text{display}; \\
\text{Window} w; \\
\text{int} * & \text{num\_prop\_return};
\end{align*}
\]

*display* Specifies the connection to the X server.

*w* Specifies the window whose property list you want to obtain.

*num\_prop\_return* Returns the length of the properties array.
The XListProperties function returns a pointer to an array of atom properties that are defined for the specified window or returns NULL if no properties were found. To free the memory allocated by this function, use XFree.

XListProperties can generate a BadWindow error.

To change a property of a given window, use XChangeProperty.

XChangeProperty(display, w, property, type, format, mode, data, nelements)

- Display *display;
- Window w;
- Atom property, type;
- int format;
- int mode;
- unsigned char *data;
- int nelements;

- display: Specifies the connection to the X server.
- w: Specifies the window whose property you want to change.
- property: Specifies the property name.
- type: Specifies the type of the property. The X server does not interpret the type but simply passes it back to an application that later calls XGetWindowProperty.
- format: Specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities. Possible values are 8, 16, and 32. This information allows the X server to correctly perform byte-swap operations as necessary. If the format is 16-bit or 32-bit, you must explicitly cast your data pointer to an (unsigned char *) in the call to XChangeProperty.
- mode: Specifies the mode of the operation. You can pass PropModeReplace, PropModePrepend, or PropModeAppend.
- data: Specifies the property data.
- nelements: Specifies the number of elements of the specified data format.

The XChangeProperty function alters the property for the specified window and causes the X server to generate a PropertyNotify event on that window. XChangeProperty performs the following:

- If mode is PropModeReplace, XChangeProperty discards the previous property value and stores the new data.
- If mode is PropModePrepend or PropModeAppend, XChangeProperty inserts the specified data before the beginning of the existing data or onto the end of the existing data, respectively. The type and format must match the existing property value, or a BadMatch error results. If the property is undefined, it is treated as defined with the correct type and format with zero-length data.

The lifetime of a property is not tied to the storing client. Properties remain until explicitly deleted, until the window is destroyed, or until the server resets. For a discussion of what happens when the connection to the X server is closed, see section 2.6. The maximum size of a property is server dependent and can vary dynamically depending on the amount of memory the server has available. (If there is insufficient space, a BadAlloc error results.)

XChangeProperty can generate BadAlloc, BadAtom, BadMatch, BadValue, and BadWindow errors.

To rotate a window’s property list, use XRotateWindowProperties.
XRotateWindowProperties(
    display, w, properties, num_prop, npositions
)

Display *display;
Window w;
Atom properties[];
int num_prop;
int npositions;

display Specifies the connection to the X server.
w Specifies the window.
properties Specifies the array of properties that are to be rotated.
num_prop Specifies the length of the properties array.
npositions Specifies the rotation amount.

The XRotateWindowProperties function allows you to rotate properties on a window and causes the X server to generate PropertyNotify events. If the property names in the properties array are viewed as being numbered starting from zero and if there are num_prop property names in the list, then the value associated with property name I becomes the value associated with property name (I + npositions) mod N for all I from zero to N - 1. The effect is to rotate the states by npositions places around the virtual ring of property names (right for positive npositions, left for negative npositions). If npositions mod N is nonzero, the X server generates a PropertyNotify event for each property in the order that they are listed in the array. If an atom occurs more than once in the list or no property with that name is defined for the window, a BadMatch error results. If a BadAtom or BadMatch error results, no properties are changed.

XRotateWindowProperties can generate BadAtom, BadMatch, and BadWindow errors.

To delete a property on a given window, use XDeleteProperty.

XDeleteProperty(
    display, w, property
)

Display *display;
Window w;
Atom property;

display Specifies the connection to the X server.
w Specifies the window whose property you want to delete.
property Specifies the property name.

The XDeleteProperty function deletes the specified property only if the property was defined on the specified window and causes the X server to generate a PropertyNotify event on the window unless the property does not exist.

XDeleteProperty can generate BadAtom and BadWindow errors.

4.5. Selections

Selections are one method used by applications to exchange data. By using the property mechanism, applications can exchange data of arbitrary types and can negotiate the type of the data. A selection can be thought of as an indirect property with a dynamic type. That is, rather than having the property stored in the X server, the property is maintained by some client (the owner). A selection is global in nature (considered to belong to the user but be maintained by clients) rather than being private to a particular window subhierarchy or a particular set of clients.

Xlib provides functions that you can use to set, get, or request conversion of selections. This allows applications to implement the notion of current selection, which requires that notification be sent to applications when they no longer own the selection. Applications that support selection often highlight the current selection and so must be informed when another
application has acquired the selection so that they can unhighlight the selection.

When a client asks for the contents of a selection, it specifies a selection target type. This target type can be used to control the transmitted representation of the contents. For example, if the selection is "the last thing the user clicked on" and that is currently an image, then the target type might specify whether the contents of the image should be sent in XY format or Z format.

The target type can also be used to control the class of contents transmitted, for example, asking for the "looks" (fonts, line spacing, indentation, and so forth) of a paragraph selection, not the text of the paragraph. The target type can also be used for other purposes. The protocol does not constrain the semantics.

To set the selection owner, use XSetSelectionOwner.

XSetSelectionOwner( display, selection, owner, time )

- display Specifies the connection to the X server.
- selection Specifies the selection atom.
- owner Specifies the owner of the specified selection atom. You can pass a window or None.
- time Specifies the time. You can pass either a timestamp or CurrentTime.

The XSetSelectionOwner function changes the owner and last-change time for the specified selection and 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 X server time. Otherwise, the last-change time is set to the specified time, with CurrentTime replaced by the current server time. If the owner window is specified as None, then the owner of the selection becomes None (that is, no owner). Otherwise, the owner of the selection becomes the client executing the request.

If the new owner (whether a client or None) is not the same as the current owner of the selection and the current owner is not None, the current owner is sent a SelectionClear event. If the client that is the owner of a selection is later terminated (that is, its connection is closed) or if the owner window it has specified in the request is later destroyed, the owner of the selection automatically reverts to None, but the last-change time is not affected. The selection atom is uninterpreted by the X server. XGetSelectionOwner returns the window ID associated with the window that currently owns the specified selection. If no selection was specified, the function returns the constant None. If None is returned, there is no owner for the selection.

XSetSelectionOwner can generate BadAtom and BadWindow errors.

To return the selection owner, use XGetSelectionOwner.

Window XGetSelectionOwner( display, selection )

- display Specifies the connection to the X server.
- selection Specifies the selection atom whose owner you want returned.

The XGetSelectionOwner function returns the window ID associated with the window that currently owns the specified selection. If no selection was specified, the function returns the constant None. If None is returned, there is no owner for the selection.

XGetSelectionOwner can generate a BadAtom error.
To request conversion of a selection, use XConvertSelection.

XConvertSelection(display, selection, target, property, requestor, time)

Display *display;
Atom selection, target;
Atom property;
Window requestor;
Time time;

display Specifies the connection to the X server.
selection Specifies the selection atom.
target Specifies the target atom.
property Specifies the property name. You also can pass None.
requestor Specifies the requestor.
time Specifies the time. You can pass either a timestamp or CurrentTime.

XConvertSelection requests that the specified selection be converted to the specified target type:

• If the specified selection has an owner, the X server sends a SelectionRequest event to that owner.
• If no owner for the specified selection exists, the X server generates a SelectionNotify event to the requestor with property None.

The arguments are passed on unchanged in either of the events. There are two predefined selection atoms: PRIMARY and SECONDARY.

XConvertSelection can generate BadAtom and BadWindow errors.
Chapter 5

Pixmap and Cursor Functions

Once you have connected to an X server, you can use the Xlib functions to:

- Create and free pixmaps
- Create, recolor, and free cursors

5.1. Creating and Freeing Pixmaps

Pixmaps can only be used on the screen on which they were created. Pixmaps are off-screen resources that are used for various operations, for example, defining cursors as tiling patterns or as the source for certain raster operations. Most graphics requests can operate either on a window or on a pixmap. A bitmap is a single bit-plane pixmap.

To create a pixmap of a given size, use \texttt{XCreatePixmap}.

\begin{verbatim}
Pixmap XCreatePixmap(display, d, width, height, depth)
  Display *display;
  Drawable d;
  unsigned int width, height;
  unsigned int depth;

  \texttt{display} \hspace{1em} Specifies the connection to the X server.
  \texttt{d} \hspace{1em} Specifies which screen the pixmap is created on.
  \texttt{width} \hspace{1em} Specify the width and height, which define the dimensions of the pixmap.
  \texttt{depth} \hspace{1em} Specifies the depth of the pixmap.
\end{verbatim}

The \texttt{XCreatePixmap} function creates a pixmap of the width, height, and depth you specified and returns a pixmap ID that identifies it. It is valid to pass an \texttt{InputOnly} window to the drawable argument. The width and height arguments must be nonzero, or a \texttt{BadValue} error results. The depth argument must be one of the depths supported by the screen of the specified drawable, or a \texttt{BadValue} error results.

The server uses the specified drawable to determine on which screen to create the pixmap. The pixmap can be used only on this screen and only with other drawables of the same depth (see \texttt{XCopyPIane} for an exception to this rule). The initial contents of the pixmap are undefined.

\texttt{XCreatePixmap} can generate \texttt{BadAlloc}, \texttt{BadDrawable}, and \texttt{BadValue} errors.

To free all storage associated with a specified pixmap, use \texttt{XFreePixmap}.

\begin{verbatim}
XFreePixmap(display, pixmap)
  Display *display;
  Pixmap pixmap;

  \texttt{display} \hspace{1em} Specifies the connection to the X server.
  \texttt{pixmap} \hspace{1em} Specifies the pixmap.
\end{verbatim}

The \texttt{XFreePixmap} function first deletes the association between the pixmap ID and the pixmap. Then, the X server frees the pixmap storage when there are no references to it. The pixmap should never be referenced again.
5.2. Creating, Recoloring, and Freeing Cursors

Each window can have a different cursor defined for it. Whenever the pointer is in a visible window, it is set to the cursor defined for that window. If no cursor was defined for that window, the cursor is the one defined for the parent window.

From X’s perspective, a cursor consists of a cursor source, mask, colors, and a hotspot. The mask pixmap determines the shape of the cursor and must be a depth of one. The source pixmap must have a depth of one, and the colors determine the colors of the source. The hotspot defines the point on the cursor that is reported when a pointer event occurs. There may be limitations imposed by the hardware on cursors as to size and whether a mask is implemented. XQueryBestCursor can be used to find out what sizes are possible. There is a standard font for creating cursors, but Xlib provides functions that you can use to create cursors from an arbitrary font, or from bitmaps.

To create a cursor from the standard cursor font, use XCreateFontCursor.

```c
#include <X11/cursorfont.h>

Cursor XCreateFontCursor(display, shape)
    Display *display;
    unsigned int shape;

display Specifies the connection to the X server.
shape Specifies the shape of the cursor.
```

X provides a set of standard cursor shapes in a special font named cursor. Applications are encouraged to use this interface for their cursors because the font can be customized for the individual display type. The shape argument specifies which glyph of the standard fonts to use.

The hotspot comes from the information stored in the cursor font. The initial colors of a cursor are a black foreground and a white background (see XRecolorCursor). For further information about cursor shapes, see appendix B.

XCreateFontCursor can generate BadAlloc and BadValue errors.

To create a cursor from font glyphs, use XCreateGlyphCursor.

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

display Specifies the connection to the X server.
source_font Specifies the font for the source glyph.
mask_font Specifies the font for the mask glyph or None.
source_char Specifies the character glyph for the source.
mask_char Specifies the glyph character for the mask.
foreground_color Specifies the RGB values for the foreground of the source.
background_color Specifies the RGB values for the background of the source.
```

XFreePixmap can generate a BadPixmap error.
The XCreateGlyphCursor function is similar to XCreatePixmapCursor except that the source and mask bitmaps are obtained from the specified font glyphs. The source_char must be a defined glyph in source_font, or a BadValue error results. If mask_font is given, mask_char must be a defined glyph in mask_font, or a BadValue error results. The mask_font and character are optional. The origins of the source_char and mask_char (if defined) glyphs are positioned coincidently and define the hotspot. The source_char and mask_char need not have the same bounding box metrics, and there is no restriction on the placement of the hotspot relative to the bounding boxes. If no mask_char is given, all pixels of the source are displayed. You can free the fonts immediately by calling XFreeFont if no further explicit references to them are to be made.

For 2-byte matrix fonts, the 16-bit value should be formed with the byte1 member in the most-significant byte and the byte2 member in the least-significant byte.

XCreateGlyphCursor can generate BadAlloc, BadFont, and BadValue errors.

To create a cursor from two bitmaps, use XCreatePixmapCursor.

Cursor XCreatePixmapCursor(display, source, mask, foreground_color, background_color, x, y)

Display *display;
Pixmap source;
Pixmap mask;
XColor *foreground_color;
XColor *background_color;
unsigned int x, y;

display Specifies the connection to the X server.
source Specifies the shape of the source cursor.
mask Specifies the cursor's source bits to be displayed or None.
foreground_color Specifies the RGB values for the foreground of the source.
background_color Specifies the RGB values for the background of the source.
x y Specify the x and y coordinates, which indicate the hotspot relative to the source's origin.

The XCreatePixmapCursor function creates a cursor and returns the cursor ID associated with it. The foreground and background RGB values must be specified using foreground_color and background_color, even if the X server only has a StaticGray or Grayscale screen. The foreground color is used for the pixels set to 1 in the source, and the background color is used for the pixels set to 0. Both source and mask, if specified, must have depth one (or a BadMatch error results) but can have any root. The mask argument defines the shape of the cursor. The pixels set to 1 in the mask define which source pixels are displayed, and the pixels set to 0 define which pixels 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 pixmap defined by the source argument, or a BadMatch error results. The hotspot must be a point within the source, or a BadMatch error results.

The components of the cursor can 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 X server might or might not make a copy of the pixmap.

XCreatePixmapCursor can generate BadAlloc and BadPixmap errors.

To determine useful cursor sizes, use XQueryBestCursor.
Status XQueryBestCursor(display, d, width, height, width_return, height_return)
    Display *display;
    Drawable d;
    unsigned int width, height;
    unsigned int *width_return, *height_return;

display Specifies the connection to the X server.
d Specifies the drawable, which indicates the screen.
width Specify the width and height of the cursor that you want the size information for.
width_return Return the best width and height that is closest to the specified width and height.
height_return

Some displays allow larger cursors than other displays. The XQueryBestCursor function provides a way to find out what size cursors are actually possible on the display. It returns the largest size that can be displayed. Applications should be prepared to use smaller cursors on displays that cannot support large ones.

XQueryBestCursor can generate a BadDrawable error.

To change the color of a given cursor, use XRecolorCursor.
XRecolorCursor(display, cursor, foreground_color, background_color)
    Display *display;
    Cursor cursor;
    XColor *foreground_color, *background_color;

display Specifies the connection to the X server.
cursor Specifies the cursor.
foreground_color Specifies the RGB values for the foreground of the source.
background_color Specifies the RGB values for the background of the source.

The XRecolorCursor function changes the color of the specified cursor, and if the cursor is being displayed on a screen, the change is visible immediately. Note that the pixel members of the XColor structures are ignored, only the RGB values are used.

XRecolorCursor can generate a BadCursor error.

To free (destroy) a given cursor, use XFreeCursor.
XFreeCursor(display, cursor)
    Display *display;
    Cursor cursor;

display Specifies the connection to the X server.
cursor Specifies the cursor.

The XFreeCursor function deletes the association between the cursor resource ID and the specified cursor. The cursor storage is freed when no other resource references it. The specified cursor ID should not be referred to again.

XFreeCursor can generate a BadCursor error.
Chapter 6

Color Management Functions

Each X window always has an associated colormap that provides a level of indirection between pixel values and colors displayed on the screen. Xlib provides functions that you can use to manipulate a colormap. The X protocol defines colors using values in the RGB color space. The RGB color space is device-dependent; rendering an RGB value on differing output devices typically results in different colors. Xlib also provides a means for clients to specify color using device-independent color spaces, for consistent results across devices. Xlib supports device-independent color spaces derivable from the CIE XYZ color space. This includes the CIE XYZ, xyY, L*u*v*, and L*a*b* color spaces as well as the TekHVC color space.

This chapter discusses how to:

- Create, copy, and destroy a colormap
- Specify colors by name or value
- Allocate, modify, and free color cells
- Read entries in a colormap
- Convert between color spaces
- Control aspects of color conversion
- Query the color gamut of a screen
- Add new color spaces

All functions, types, and symbols in this chapter with the prefix "Xcms" are defined in <X11/Xcms.h>. The remaining functions and types are defined in <X11/Xlib.h>.

Functions in this chapter manipulate the representation of color on the screen. For each possible value that a pixel can take in a window, there is a color cell in the colormap. For example, if a window is 4 bits deep, pixel values 0 through 15 are defined. A colormap is a collection of color cells. A color cell consists of a triple of red, green, and blue values. The hardware imposes limits on the number of significant bits in these values. As each pixel is read out of display memory, the pixel is looked up in a colormap. The RGB value of the cell determines what color is displayed on the screen. On a grayscale display with a black-and-white monitor the values are combined to determine the brightness on the screen.

Typically, an application allocates color cells or sets of color cells to obtain the desired colors. The client can allocate read-only cells, in which case the pixel values for these colors can be shared among multiple applications, and the RGB value of the cell cannot be changed. If the client allocates read/write cells, they are exclusively owned by the client, and the color associated with the pixel value may be changed at will. Cells must be allocated (and, if read/write, initialized with an RGB value) by a client to obtain desired colors; use of pixel value for an unallocated cell results in an undefined color.

Because colormaps are associated with windows, X supports displays with multiple colormaps and, indeed, different types of colormaps. If there are not sufficient colormap resources in the display, some windows will display in their true colors, and others will display with incorrect colors. A window manager usually controls which windows are displayed in their true colors if more than one colormap is required for the color resources the applications are using. At any time, there is a set of installed colormaps for a screen. Windows using one of the installed colormaps display with true colors, and windows using other colormaps generally display with incorrect colors. The set of installed colormaps is controlled using XInstallColormap and XUninstallColormap.
Colormaps are local to a particular screen. Screens always have a default colormap, and programs typically allocate cells out of this colormap. You should not in general write applications that monopolize color resources. Although some hardware supports multiple colormaps installed at one time, many of the hardware displays built today support only a single installed colormap, so the primitives are written to encourage sharing of colormap entries between applications.

The DefaultColormap macro returns the default colormap. The DefaultVisual macro returns the default visual type for the specified screen. Possible visual types are StaticGray, Grayscale, StaticColor, PseudoColor, TrueColor, or DirectColor (see section 3.1).

## 6.1. Color Structures

Functions which operate only on RGB color space values use an XColor structure, which contains:

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

The red, green, and blue values are always in the range 0 to 65535 inclusive, independent of the number of bits actually used in the display hardware. The server scales these values down to the range used by the hardware. Black is represented by (0,0,0), white is represented by (65535,65535,65535). In some functions, the flags member controls which of the red, green, and blue members is used and can be the inclusive OR of zero or more of DoRed, DoGreen, and DoBlue.

Functions which operate on all color space values use an XcmsColor structure. This structure contains a union of substructures, each supporting color specification encoding for a particular color space. Like the XColor structure, the XcmsColor structure contains pixel and color specification information (the spec member in the XcmsColor structure).

```c
typedef unsigned long XcmsColorFormat; /* Color Specification Format */
typedef struct {
    union {
        XcmsRGB RGB;
        XcmsRGBi RGBi;
        XcmsCIEXYZ CIEXYZ;
        XcmsCIEuvY CIEuvY;
        XcmsCIExyY CIExyY;
        XcmsCIELab CIELab;
        XcmsCIELuv CIELuv;
        XcmsTekHVC TekHVC;
        XcmsPad Pad;
    } spec;
    XcmsColorFormat format;
    unsigned long pixel;
} XcmsColor;  /* Xcms Color Structure */
```

Because the color specification can be encoded for the various color spaces, encoding for the spec member is identified by the format member, which is of type XcmsColorFormat. The following macros define standard formats.
#define XcmsUndefinedFormat 0x00000000
#define XcmsCIEXYZFormat 0x00000001 /* CIE XYZ */
#define XcmsCIEuvYFormat 0x00000002 /* CIE u'v'Y */
#define XcmsCIExyYFormat 0x00000003 /* CIE xyY */
#define XcmsCIELabFormat 0x00000004 /* CIE L*a*b* */
#define XcmsCIELuvFormat 0x00000005 /* CIE L*u*v* */
#define XcmsTekH VC Format 0x00000006 /* TekHVC */
#define XcmsRGBFormat 0x80000000 /* RGB Device */
#define XcmsRGBiFormat 0x80000001 /* RGB Intensity */

Note that formats for device-independent color spaces are distinguishable from those for
device-dependent spaces by the 32nd bit. If this bit is set, it indicates that the color
specification is in a device-dependent form; otherwise, it is in a device-independent form. If
the 31st bit is set, this indicates that the color space has been added to Xlib at run-time (see
section 6.12.4). The format value for a color space added at run-time may be different each
time the program is executed. If references to such a color space must be made outside the
client (for example, storing a color specification in a file), then reference should be made by
color space string prefix (see XcmsFormatOfPrefix and XcmsPrefixOfFormat).

Data types that describe the color specification encoding for the various color spaces are
defined as follows:

typedef double XcmsFloat;

typedef struct {
    unsigned short red; /* 0x0000 to 0xffff */
    unsigned short green; /* 0x0000 to 0xffff */
    unsigned short blue; /* 0x0000 to 0xffff */
} XcmsRGB; /* RGB Device */

typedef struct {
    XcmsFloat red; /* 0.0 to 1.0 */
    XcmsFloat green; /* 0.0 to 1.0 */
    XcmsFloat blue; /* 0.0 to 1.0 */
} XcmsRGBi; /* RGB Intensity */

typedef struct {
    XcmsFloat X; /* 0.0 to 1.0 */
    XcmsFloat Y; /* 0.0 to 1.0 */
    XcmsFloat Z; /* 0.0 to 1.0 */
} XcmsCIEXYZ; /* CIE XYZ */

typedef struct {
    XcmsFloat u_prime; /* 0.0 to -.6 */
    XcmsFloat v_prime; /* 0.0 to -.6 */
    XcmsFloat Y; /* 0.0 to 1.0 */
} XcmsCIEuvY; /* CIE u'v'Y */

typedef struct {
    XcmsFloat x; /* 0.0 to -.75 */
    XcmsFloat y; /* 0.0 to -.85 */
    XcmsFloat Y; /* 0.0 to 1.0 */
} XcmsCIExyY; /* CIE xyY */

typedef struct {
    XcmsFloat L_star; /* 0.0 to 100.0 */
    XcmsFloat a_star;
}
typedef struct {
    XcmsFloat L_star; /* 0.0 to 100.0 */
    XcmsFloat u__star;
    XcmsFloat v__star;
} XcmsCIELab; /* CIE L*a*b* */

typedef struct {
    XcmsFloat H; /* 0.0 to 360.0 */
    XcmsFloat V; /* 0.0 to 100.0 */
    XcmsFloat C; /* 0.0 to 100.0 */
} XcmsCIELuv; /* CIE L*u*v* */

typedef struct {
    XcmsFloat pad0;
    XcmsFloat pad1;
    XcmsFloat pad2;
    XcmsFloat pad3;
} XcmsPad; /* four doubles */

The device-dependent formats provided allow color specification in:

- **RGB Intensity (XcmsRGBi)**
  
  Red, green, and blue linear intensity values, floating point values from 0.0 to 1.0, where 1.0 indicates full intensity, 0.5 half intensity, and so on.

- **RGB Device (XcmsRGB)**
  
  Red, green, and blue values appropriate for the specified output device. XcmsRGB values are of type unsigned short, scaled from 0 to 65535 inclusive, and are interchangeable with values the red, green, and blue values in an XColor structure.

It is important to note that RGB Intensity values are not gamma corrected values. In contrast, RGB Device values generated as a result of converting color specifications are always gamma corrected, and RGB Device values acquired as a result of querying a colormap or passed in by the client are assumed by Xlib to be gamma corrected. The term "RGB value" in this manual always refers to an RGB Device value.

### 6.2. Color Strings

Xlib provides a mechanism for using string names for colors. A color string may either contain an abstract color name or a numerical color specification. Color strings are case-insensitive.

Color strings are used in the following functions:

- `XAllocNamedColor`
- `XcmsAllocateNamedColor`
- `XLookupColor`
- `XcmsLookupColor`
- `XParseColor`
- `XStoreNamedColor`

Xlib supports the use of abstract color names, for example, "red", "blue". A value for this abstract name is obtained by searching one or more color name databases. Xlib first searches zero or more client-side databases; the number, location, and content of these databases is implementation dependent, and might depend on the current locale. If the name is not found,
Xlib then looks for the color in the X server's database. If the color name is not in the Host Portable Character Encoding the result is implementation dependent.

A numerical color specification consists of a color space name and a set of values in the following syntax:

\<color_space_name>:<value>/.../<value>\

The following are examples of valid color strings.

"CIEXYZ:0.3227/0.28133/0.2493"
"RGBi:1.0/0.0/0.0"
"rgb:00/ff/00"
"CIELuv:50.0/0.0/0.0"

The syntax and semantics of numerical specifications are given for each standard color space in sections below.

6.2.1. RGB Device String Specification

An RGB Device specification is identified by the prefix "rgb:" and conforms to the following syntax:

rgb:<red>/<green>/<blue>

\<red>, <green>, <blue> := h | hh | hhh | hhhh

\h := single hexadecimal digits (case insignificant)\

Note that \h indicates the value scaled in 4 bits, hh the value scaled in 8 bits, hhh the value scaled in 12 bits, and hhhh the value scaled in 16 bits, respectively.

Typical examples are "rgb:ca/75/52" and "rgb:ccc/320/320", but mixed numbers of hex digits ("rgb:ff/a5/0" and "rgb:ccc/32/0") are also allowed.

For backward compatibility, an older syntax for RGB Device is supported, but its continued use is not encouraged. The syntax is an initial sharp sign character followed by a numeric specification, in one of the following formats:

#RGB (4 bits each)
#RRGGBB (8 bits each)
#RRRGGGBBB (12 bits each)
#RRRRGGGGBBBB (16 bits each)

The R, G, and B represent single hexadecimal digits. When fewer than 16 bits each are specified, they represent the most-significant bits of the value (unlike the "rgb:" syntax, in which values are scaled). For example, #3a7 is the same as #3000a0007000.

6.2.2. RGB Intensity String Specification

An RGB intensity specification is identified by the prefix "rgbi:" and conforms to the following syntax:

rgbi:<red>/<green>/<blue>

Note that red, green, and blue are floating point values between 0.0 and 1.0, inclusive. The input format for these values is an optional sign, a string of numbers possibly containing a decimal point, and an optional exponent field containing an E or e followed by a possibly signed integer string.

6.2.3. Device-Independent String Specifications

The standard device-independent string specifications have the following syntax:
All of the values (C, H, V, X, Y, Z, a, b, u, v, y, x) are floating point values. The syntax for these values is an optional '+' or '-' sign, a string of digits possibly containing a decimal point, and an optional exponent field consisting of an 'E' or 'e' followed by an optional '+' or '-' followed by a string of digits.

6.3. Color Conversion Contexts and Gamut Mapping

When Xlib converts device-independent color specifications into device-dependent specifications and vice-versa, it uses knowledge about the color limitations of the screen hardware. This information, typically called the device profile, is available in a Color Conversion Context (hereafter referred to as the CCC).

Because a specified color may be outside the color gamut of the target screen and the white point associated with the color specification may differ from the white point inherent to the screen, Xlib applies gamut mapping when certain conditions are encountered:

- Gamut compression when conversion of device-independent color specifications to device-dependent color specification results in a color out of the target screen's gamut.
- White adjustment when the inherent white point of the screen differs from the white point assumed by the client.

Gamut handling methods are stored as callbacks in the CCC, which, in turn, are used by the color space conversion routines. Client data is also stored in the CCC for each callback. The CCC also contains the white point the client assumes to be associated with color specifications (that is, the Client White Point). The client can specify the gamut handling callbacks and client data, as well as the Client White Point. Note that Xlib does not preclude the X client from performing other forms of gamut handling (for example, gamut expansion); however, direct support for gamut handling other than white adjustment and gamut compression is not provided by Xlib.

Associated with each colormap is an initial CCC transparently generated by Xlib. Therefore, when you specify a colormap as an argument to an Xlib function, you are indirectly specifying a CCC. There is a default CCC associated with each screen. Newly created CCCs inherit attributes from the default CCC, so the default CCC attributes can be modified to affect new CCCs.

Xcms functions in which gamut mapping can occur return Status, and have specific status values defined for them:

- XcmsFailure indicates that the function failed.
- XcmsSuccess indicates that the function succeeded. In addition, if the function performed any color conversion, the color (or colors) did not need to be compressed.
- XcmsSuccessWithCompression indicates the function performed color conversion, and at least one of the colors needed to be compressed. The gamut compression method is determined by the gamut compression procedure in the CCC that is specified directly as a function argument, or in the CCC indirectly specified by means of the colormap argument.

6.4. Creating, Copying, and Destroying Colormaps

To create a colormap for a screen, use XCreateColormap.
Colormap XCreateColormap(display, w, visual, alloc)
  Display *display;
  Window w;
  Visual *visual;
  int alloc;

  display  Specifies the connection to the X server.
  w  Specifies the window on whose screen you want to create a colormap.
  visual  Specifies a visual type supported on the screen. If the visual type is not one
           supported by the screen, a BadMatch error results.
  alloc  Specifies the colormap entries to be allocated. You can pass AllocNone or
          AllocAll.

The XCreateColormap function creates a colormap of the specified visual type for the screen
on which the specified window resides and returns the colormap ID associated with it. Note
that the specified window is only used to determine the screen.

The initial values of the colormap entries are undefined for the visual classes Grayscale,
PseudoColor, and DirectColor. For StaticGray, StaticColor, and TrueColor, the entries
have defined values, but those values are specific to the visual and are not defined by X. For
StaticGray, StaticColor, and TrueColor, alloc must be AllocNone, or a BadMatch error
results. For the other visual classes, if alloc is AllocNone, the colormap initially has no allo¬
cated entries, and clients can allocate them. For information about the visual types, see section
3.1.

If alloc is AllocAll, the entire colormap is allocated writable. The initial values of all allo¬
cated entries are undefined. For Grayscale and PseudoColor, the effect is as if an XAlloc-
ColorCells call 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 XAllocColorPlanes call
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 by using XFreeColors.

XCreateColormap can generate BadAlloc, BadMatch, BadValue, and BadWindow errors.

To create a new colormap when the allocation out of a previously shared colormap has failed
because of resource exhaustion, use XCopyColormapAndFree.

Colormap XCopyColormapAndFree(display, colormap)
  Display *display;
  Colormap colormap;

  display  Specifies the connection to the X server.
  colormap  Specifies the colormap.

The XCopyColormapAndFree function creates a colormap of the same visual type and for
the same screen as the specified colormap and returns the new colormap ID. It also moves all
of the client’s existing allocation from the specified colormap to the new colormap with their
color values intact and their read-only or writable characteristics intact and frees those entries
in the specified colormap. Color values in other entries in the new colormap are undefined. If
the specified colormap was created by the client with alloc set to AllocAll, the new colormap
is also created with AllocAll, all color values for all entries are copied from the specified
colormap, and then all entries in the specified colormap are freed. If the specified colormap
was not created by the client with AllocAll, the allocations to be moved are all those pixels
and planes that have been allocated by the client using XAllocColor, XAllocNamedColor,
XAllocColorCells, or XAllocColorPlanes and that have not been freed since they were allo¬
located.
XCopyColormapAndFree can generate BadAlloc and BadColor errors.

To destroy a colormap, use XFreeColormap.

XFreeColormap(display, colormap)
   Display *display;
   Colormap colormap;

*display Specifies the connection to the X server.
*colormap Specifies the colormap that you want to destroy.

The XFreeColormap function deletes the association between the colormap resource ID and the colormap and frees the colormap storage. However, this function has no effect on the default colormap for a screen. If the specified colormap is an installed map for a screen, it is uninstalled (see XUninstallColormap). If the specified colormap is defined as the colormap for a window (by XCreateWindow, XSetWindowColormap, or XChangeWindowAttributes), XFreeColormap changes the colormap associated with the window to None and generates a ColormapNotify event. X does not define the colors displayed for a window with a colormap of None.

XFreeColormap can generate a BadColor error.

6.5. Mapping Color Names to Values

To map a color name to an RGB value, use XLookupColor.

Status XLookupColor(display, colormap, color_name, exact_def_return, screen_def_return)
   Display *display;
   Colormap colormap;
   char *color_name;
   XColor *exact_def_return, *screen_def_return;

*display Specifies the connection to the X server.
*colormap Specifies the colormap.
*color_name Specifies the color name string (for example, red) whose color definition structure you want returned.
*exact_def_return Returns the exact RGB values.
*screen_def_return Returns the closest RGB values provided by the hardware.

The XLookupColor function looks up the string name of a color with respect to the screen associated with the specified colormap. It returns both the exact color values and the closest values provided by the screen with respect to the visual type of the specified colormap. If the color name is not in the Host Portable Character Encoding the result is implementation dependent. Use of uppercase or lowercase does not matter. XLookupColor returns nonzero if the name is resolved, otherwise it returns zero.

XLookupColor can generate a BadColor error.

To map a color name to just the exact RGB value, use XParseColor.

Status XParseColor(display, colormap, spec, exact_def_return)
   Display *display;
   Colormap colormap;
   char *spec;
   XColor *exact_def_return;

69
display Specifies the connection to the X server.
colormap Specifies the colormap.
spec Specifies the color name string; case is ignored.
exact_def_return Returns the exact color value for later use and sets the DoRed, DoGreen, and DoBlue flags.

The XParseColor function looks up the string name of a color with respect to the screen associated with the specified colormap. It returns the exact color value. If the color name is not in the Host Portable Character Encoding the result is implementation dependent. Use of uppercase or lowercase does not matter. XParseColor returns nonzero if the name is resolved, otherwise it returns zero.

XParseColor can generate a BadColor error.

To map a color name to a value in an arbitrary color space, use XcmsLookupColor.

Status XcmsLookupColor(display, colormap, color_string, color_exact_return, color_screen_return, result_format)

Display *display;
Colormap colormap;
char *color_string;
XcmsColor *color_exact_return, *color_screen_return;
XcmsColorFormat result_format;
display Specifies the connection to the X server.
colormap Specifies the colormap.
color_string Specifies the color string.
color_exact_return Returns the color specification parsed from the color string or parsed from the corresponding string found in a color name database.
color_screen_return Returns the color that can be reproduced on the Screen.
result_format Specifies the color format for the returned color specifications (color_screen_return and color_exact_return arguments). If format is XcmsUndefinedFormat and the color string contains a numerical color specification, the specification is returned in the format used in that numerical color specification. If format is XcmsUndefinedFormat and the color string contains a color name, the specification is returned in the format used to store the color in the database.

The XcmsLookupColor function looks up the string name of a color with respect to the screen associated with the specified colormap. It returns both the exact color values and the closest values provided by the screen with respect to the visual type of the specified colormap. The values are returned in the format specified by result_format. If the color name is not in the Host Portable Character Encoding the result is implementation dependent. Use of uppercase or lowercase does not matter. XcmsLookupColor returns XcmsSuccess or XcmsSuccessWithCompression if the name is resolved, otherwise it returns XcmsFailure. If XcmsSuccessWithCompression is resumed, then the color specification in color_screen_return is the result of gamut compression.

6.6. Allocating and Freeing Color Cells

There are two ways of allocating color cells: explicitly as read-only entries, one pixel value at a time, or read/write, where you can allocate a number of color cells and planes simultaneously. A read-only cell has its RGB value set by the server. Read/write cells do not have defined colors initially; functions described in the next section must be used to store values.
into them. Although it is possible for any client to store values into a read/write cell allocated by another client, read/write cells normally should be considered private to the client that allocated them.

Read-only colormap cells are shared among clients. The server counts each allocation and free of the cell by clients. When the last client frees a shared cell, the cell is finally deallocated. Note that if a single client allocates the same read-only cell multiple times, the server counts each such allocation, not just the first one.

To allocate a read-only color cell with an RGB value, use XAllocColor.

Status XAllocColor(display, colormap, screen_in_out)

- Display *display;
- Colormap colormap;
- XColor *screen_in_out;

- display Specifies the connection to the X server.
- colormap Specifies the colormap.
- screen_in_out Specifies and returns the values actually used in the colormap.

The XAllocColor function allocates a read-only colormap entry corresponding to the closest RGB value supported by the hardware. XAllocColor returns the pixel value of the color closest to the specified RGB elements supported by the hardware and returns the RGB value actually used. The corresponding colormap cell is read-only. In addition, XAllocColor returns nonzero if it succeeded or zero if it failed. Multiple clients that request the same effective RGB value can be assigned the same read-only entry, thus allowing entries to be shared. When the last client deallocates a shared cell, it is deallocated. XAllocColor does not use or affect the flags in the XColor structure.

XAllocColor can generate a BadColor error.

To allocate a read-only color cell with a color in arbitrary format, use XcmsAllocColor.

Status XcmsAllocColor(display, colormap, color_in_out, result_format)

- Display *display;
- Colormap colormap;
- XcmsColor *color_in_out;
- XcmsColorFormat result_format;

- display Specifies the connection to the X server.
- colormap Specifies the colormap.
- color_in_out Specifies the color to allocate and returns the pixel and color that is actually used in the colormap.
- result_format Specifies the color format for the returned color specification.

The XcmsAllocColor function is similar to XAllocColor except the color can be specified in any format. The XcmsAllocColor function ultimately calls XAllocColor to allocate a read-only color cell (colormap entry) with the specified color. XcmsAllocColor first converts the color specified to an RGB value and then passes this to XAllocColor. XcmsAllocColor returns the pixel value of the color cell and the color specification actually allocated. This returned color specification is the result of converting the RGB value returned by XAllocColor into the format specified with the result_format argument. If there is no interest in a returned color specification, unnecessary computation can be bypassed if result_format is set to XcmsRGBFormat. The corresponding colormap cell is read-only. If this routine returns XcmsFailure, the color_in_out color specification is left unchanged.

XcmsAllocColor can generate a BadColor error.
To allocate a read-only color cell using a color name, and return the closest color supported by
the hardware in RGB format, use XAllocNamedColor.

Status XAllocNamedColor(display, colormap, color_name, screen_def_return, exact_def_return)
    Display *display;
    Colormap colormap;
    char *color_name;
    XColor *screen_def_return, *exact_def_return;

display Specifies the connection to the X server.
colormap Specifies the colormap.
color_name Specifies the color name string (for example, red) whose color definition struc-
ture you want returned.

screen_def_return Returns the closest RGB values provided by the hardware.
exact_def_return Returns the exact RGB values.

The XAllocNamedColor function looks up the named color with respect to the screen that is
associated with the specified colormap. It returns both the exact database definition and the
closest color supported by the screen. The allocated color cell is read-only. The pixel value is
returned in screen_def_return. If the color name is not in the Host Portable Character Encod-
ing the result is implementation dependent. Use of uppercase or lowercase does not matter.
XLookupColor returns nonzero if a cell is allocated, otherwise it returns zero.

XAllocNamedColor can generate a BadColor error.

To allocate a read-only color cell using a color name, and return the closest color supported by
the hardware in an arbitrary format, use XcmsAllocNamedColor.

Status XcmsAllocNamedColor(display, colormap, color_string, result_format, color_screen_return,
color_exact_return)
    Display *display;
    Colormap colormap;
    char *color_string;
    XcmsColorFormat result_format;
    XcmsColor *color_screen_return;
    XcmsColor *color_exact_return;

display Specifies the connection to the X server.
colormap Specifies the colormap.
color_string Specifies the color string whose color definition structure is to be returned.
result_format Specifies the color format for the returned color specifications
    (color_screen_return and color_exact_return arguments). If format is
    XcmsUndefinedFormat and the color string contains a numerical color
    specification, the specification is returned in the format used in that numerical
    color specification. If format is XcmsUndefinedFormat and the color string
    contains a color name, the specification is returned in the format used to store
    the color in the database.

color_screen_return Returns the pixel value of the color cell and color specification that actually is
    stored for that cell.

color_exact_return Returns the color specification parsed from the color string or parsed from the
    corresponding string found in a color name database.
The XcmsAHocNamedColor function is similar to XAllocNamedColor except the color returned can be in any format specified. This function ultimately calls XAllocColor to allocate a read-only color cell with the color specified by a color string. The color string is parsed into an XcmsColor structure (see XcmsLookupColor), converted to an RGB value, then finally passed to the XAllocColor. If the color name is not in the Host Portable Character Encoding the result is implementation dependent. Use of uppercase or lowercase does not matter.

This function returns both the color specification as a result of parsing (exact specification) and the actual color specification stored (screen specification). This screen specification is the result of converting the RGB value returned by XAllocColor into the format specified in result_format. If there is no interest in a returned color specification, unnecessary computation can be bypassed if result_format is set to XcmsRGBFormat.

XcmsAHocNamedColor can generate a BadColor error.

To allocate read/write color cell and color plane combinations for a PseudoColor model, use XAllocColorCells.

Status XAllocColorCells(display, colormap, contig, plane_masks_return, nplanes, pixels_return, npixels)

Display *display;
Colormap colormap;
Bool contig;
unsigned long plane_masks_return[];
unsigned int nplanes;
unsigned long pixels_return[];
unsigned int npixels;

display Specifies the connection to the X server.
colormap Specifies the colormap.
contig Specifies a Boolean value that indicates whether the planes must be contiguous.
plane_mask_return Returns an array of plane masks.
nplanes Specifies the number of plane masks that are to be returned in the plane masks array.
pixels_return Returns an array of pixel values.
npixels Specifies the number of pixel values that are to be returned in the pixels_return array.

The XAllocColorCells function allocates read/write color cells. The number of colors must be positive and the number of planes nonnegative, or a BadValue error results. If ncolors and nplanes are requested, then ncolors pixels and nplane plane masks are returned. No mask will have any bits set to 1 in common with any other mask or with any of the pixels. By ORing together each pixel with zero or more masks, ncolors * 2nplanes distinct pixels can be produced. All of these are allocated writable by the request. For GrayScale or PseudoColor, each mask has exactly one bit set to 1. For DirectColor, each has exactly three bits set to 1. If contig is True and if all masks are ORed together, a single contiguous set of bits set to 1 will be formed for GrayScale or PseudoColor and three contiguous sets of bits set to 1 (one within each pixel subfield) for DirectColor. The RGB values of the allocated entries are undefined. XAllocColorCells returns nonzero if it succeeded or zero if it failed.

XAllocColorCells can generate BadColor and BadValue errors.

To allocate read/write color resources for a DirectColor model, use XAllocColorPlanes.
Status XAllocColorPlanes(display, colormap, contig, pixels_return, ncolors, nreds, ngreens, nblues, rmask_return, gmask_return, bmask_return)

Display *display;
Colormap colormap;
Bool contig;
unsigned long pixels_return[];
int ncolors;
int nreds, ngreens, nblues;
unsigned long *rmask_return, *gmask_return, *bmask_return;

Specifies the connection to the X server.

Specifies the colormap.

Specifies a Boolean value that indicates whether the planes must be contiguous.

Returns an array of pixel values. XAllocColorPlanes returns the pixel values in this array.

Specifies the number of pixel values that are to be returned in the pixels_return array.

Specify the number of red, green, and blue planes. The value you pass must be nonnegative.

Return bit masks for the red, green, and blue planes.

The specified ncolors must be positive; and nreds, ngreens, and nblues must be nonnegative, or a BadValue error results. If ncolors colors, nreds reds, ngreens greens, and nblues blues are requested, ncolors pixels are returned; and the masks have nreds, ngreens, and nblues bits set to 1, respectively. If contig is True, each mask will have a contiguous set of bits set to 1. No mask will have any bits set to 1 in common with any other mask or with any of the pixels. For DirectColor, each mask will lie within the corresponding pixel subfield. By ORing together subsets of masks with each pixel value, ncolors * 2^(nreds + ngreens + nblues) distinct pixel values can be produced. All of these are allocated by the request. However, in the colormap, there are only ncolors * 2^nreds independent red entries, ncolors * 2^ngreens independent green entries, and ncolors * 2^nblues independent blue entries. This is true even for PseudoColor. When the colormap entry of a pixel value is changed (using XStoreColors, XStoreColor, or XStoreNamedColor), the pixel is decomposed according to the masks, and the corresponding independent entries are updated. XAllocColorPlanes returns nonzero if it succeeded or zero if it failed.

XAllocColorPlanes can generate BadColor and BadValue errors.

To free colormap cells, use XFreeColors.

XFreeColors(display, colormap, pixels, npixels, planes)

Display *display;
Colormap colormap;
unsigned long pixels[];
int npixels;
unsigned long planes;

Specifies the connection to the X server.

Specifies the colormap.
pixels Specifies an array of pixel values that map to the cells in the specified colormap.

npixels Specifies the number of pixels.

planes Specifies the planes you want to free.

The XFreeColors function frees the cells represented by pixels whose values are in the pixels array. The planes argument should not have any bits set to 1 in common with any of the pixels. The set of all pixels is produced by ORing together subsets of the planes argument with the pixels. The request frees all of these pixels that were allocated by the client (using XAllocColor, XAllocNamedColor, XAllocColorCells, and XAllocColorPlanes). Note that freeing an individual pixel obtained from XAllocColorPlanes may not actually allow it to be reused until all of its related pixels are also freed. 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 the colormap are freed, even if one or more pixels produce an error. If a specified pixel is not a valid index into the colormap, a BadValue error results. If a specified pixel is not allocated by the client (that is, is unallocated or is only allocated by another client), or if the colormap was created with all entries writable (by passing AllocAll to XCreateColormap), a BadAccess error results. If more than one pixel is in error, the one that gets reported is arbitrary.

XFreeColors can generate BadAccess, BadColor, and BadValue errors.

6.7. Modifying and Querying Colormap Cells

To store an RGB value in a single colormap cell, use XStoreColor:

XStoreColor(display, colormap, color)

Display *display;
Colormap colormap;
XColor *color;

display Specifies the connection to the X server.

colormap Specifies the colormap.

color Specifies the pixel and RGB values.

The XStoreColor function changes the colormap entry of the pixel value specified in the pixel member of the XColor structure. You specified this value in the pixel member of the XColor structure. This pixel value must be a read/write cell and a valid index into the colormap. If a specified pixel is not a valid index into the colormap, a BadValue error results. XStoreColor also changes the red, green, and/or blue color components. You specify which color components are to be changed by setting DoRed, DoGreen, and/or DoBlue in the flags member of the XColor structure. If the colormap is an installed map for its screen, the changes are visible immediately.

XStoreColor can generate BadAccess, BadColor, and BadValue errors.

To store multiple RGB values into multiple colormap cells, use XStoreColors:

XStoreColors(display, colormap, color, ncolors)

Display *display;
Colormap colormap;
XColor color[];
int ncolors;

display Specifies the connection to the X server.
colormap
Specifies the colormap.

color
Specifies an array of color definition structures to be stored.

colors
Specifies the number of XColor structures in the color definition array.

The XStoreColors function changes the colormap entries of the pixel values specified in the pixel members of the XColor structures. You specify which color components are to be changed by setting DoRed, DoGreen, and/or DoBlue in the flags member of the XColor structures. If the colormap is an installed map for its screen, the changes are visible immediately. XStoreColors changes the specified pixels if they are allocated writable in the colormap by any client, even if one or more pixels generates an error. If a specified pixel is not a valid index into the colormap, a BadValue error results. If a specified pixel either is unallocated or is allocated read-only, a BadAccess error results. If more than one pixel is in error, the one that gets reported is arbitrary.

XStoreColors can generate BadAccess, BadColor, and BadValue errors.

To store a color of arbitrary format in a single colormap cell, use XcmsStoreColor.

Status XcmsStoreColor(display, colormap, color)
   Display *display;
   Colormap colormap;
   XcmsColor *color;

   display Specifies the connection to the X server.
   colormap Specifies the colormap.
   color Specifies the color cell and the color to store. Values specified in this XcmsColor structure remain unchanged upon return.

The XcmsStoreColor function converts the color specified in the XcmsColor structure into RGB values and then uses this RGB specification in an XColor structure, whose three flags (DoRed, DoGreen, and DoBlue) are set, in a call to XStoreColor to change the color cell specified by the pixel member of the XcmsColor structure. This pixel value must be a valid index for the specified colormap, and the color cell specified by the pixel value must be a read/write cell. If the pixel value is not a valid index, a BadValue error results. If the color cell is unallocated or is allocated read-only, a BadAccess error results. If the colormap is an installed map for its screen, the changes are visible immediately.

Note that XStoreColor has no return value; therefore, a XcmsSuccess return value from this function indicates that the conversion to RGB succeeded and the call to XStoreColor was made. To obtain the actual color stored, use XcmsQueryColor. Due to the screen's hardware limitations or gamut compression, the color stored in the colormap may not be identical to the color specified.

XcmsStoreColor can generate BadAccess, BadColor, and BadValue errors.

To store multiple colors of arbitrary format into multiple colormap cells, use XcmsStoreColors.

Status XcmsStoreColors(display, colormap, colors, ncolors, compression_flags_return)
   Display *display;
   Colormap colormap;
   XcmsColor colors[];
   int ncolors;
   Bool compression_flags_return[];

   display Specifies the connection to the X server.
   colormap Specifies the colormap.
colors

Specifies the color specification array of XcmsColor structures, each specifying a color cell and the color to store in that cell. Values specified in the array remain unchanged upon return.

ncolors

Specifies the number of XcmsColor structures in the color specification array.

compression_flags_return

Specifies an array of Boolean values for returning compression status. If a non-NULL pointer is supplied, each element of the array is set to True if the corresponding color was compressed, and False otherwise. Pass NULL if the compression status is not useful.

The XcmsStoreColors function converts the colors specified in the array of XcmsColor structures into RGB values and then uses these RGB specifications in an XColor structures, whose three flags (DoRed, DoGreen, and DoBlue) are set, in a call to XStoreColors to change the color cells specified by the pixel member of the corresponding XcmsColor structure. Each pixel value must be a valid index for the specified colormap, and the color cell specified by each pixel value must be a read/write cell. If a pixel value is not a valid index, a BadValue error results. If a color cell is unallocated or is allocated read-only, a BadAccess error results. If more than one pixel is in error, the one that gets reported is arbitrary. If the colormap is an installed map for its screen, the changes are visible immediately.

Note that XStoreColors has no return value; therefore, a XcmsSuccess return value from this function indicates that conversions to RGB succeeded and the call to XStoreColors was made. To obtain the actual colors stored, use XcmsQueryColors. Due to the screen’s hardware limitations or gamut compression, the colors stored in the colormap may not be identical to the colors specified.

XcmsStoreColors can generate BadAccess, BadColor, and BadValue errors.

To store a color specified by name in a single colormap cell, use XStoreNamedColor.

XStoreNamedColor(display, colormap, color, pixel, flags)

Display *display;
Colormap colormap;
char *color;
unsigned long pixel;
int flags;

display

Specifies the connection to the X server.
colormap

Specifies the colormap.
color

Specifies the color name string (for example, red).
pixel

Specifies the entry in the colormap.
flags

Specifies which red, green, and blue components are set.

The XStoreNamedColor function looks up the named color with respect to the screen associated with the colormap and stores the result in the specified colormap. The pixel argument determines the entry in the colormap. The flags argument determines which of the red, green, and blue components are set. You can set this member to the bitwise inclusive OR of the bits DoRed, DoGreen, and DoBlue. If the color name is not in the Host Portable Character Encoding the result is implementation dependent. Use of uppercase or lowercase does not matter. If the specified pixel is not a valid index into the colormap, a BadValue error results. If the specified pixel either is unallocated or is allocated read-only, a BadAccess error results.

XStoreNamedColor can generate BadAccess, BadColor, BadName, and BadValue errors.

The XQueryColor and XQueryColors functions take pixel values in the pixel member of XColor structures, and store in the structures the RGB values for those pixels from the specified colormap. The values returned for an unallocated entry are undefined. These functions also set the flags member in the XColor structure to all three colors. If a pixel is not a
valid index into the specified colormap, a BadValue error results. If more than one pixel is in error, the one that gets reported is arbitrary.

To query the RGB value of a single colormap cell, use XQueryColor.

XQueryColor(display, colormap, def_in_out)
  Display *display;
  Colormap colormap;
  XColor *def_in_out;

display Specifies the connection to the X server.
colormap Specifies the colormap.
def_in_out Specifies and returns the RGB values for the pixel specified in the structure.

The XQueryColor function returns the current RGB value for the pixel in the XColor structure and sets the DoRed, DoGreen, and DoBlue flags.

XQueryColor can generate BadColor and BadValue errors.

To query the RGB values of multiple colormap cells, use XQueryColors.

XQueryColors(display, colormap, defs_in_out, ncolors)
  Display *display;
  Colormap colormap;
  XColor defs_in_out[];
  int ncolors;

display Specifies the connection to the X server.
colormap Specifies the colormap.
defs_in_out Specifies and returns an array of color definition structures for the pixel specified in the structure.
ncolors Specifies the number of XColor structures in the color definition array.

The XQueryColors function returns the RGB value for each pixel in each XColor structure, and sets the DoRed, DoGreen, and DoBlue flags in each structure.

XQueryColors can generate BadColor and BadValue errors.

To query the color of a single colormap cell in an arbitrary format, use XcmsQueryColor.

Status XcmsQueryColor(display, colormap, color_in_out, result_format)
  Display *display;
  Colormap colormap;
  XcmsColor *color_in_out;
  XcmsColorFormat result_format;

display Specifies the connection to the X server.
colormap Specifies the colormap.
color_in_out Specifies the pixel member that indicates the color cell to query, and the color specification stored for the color cell is returned in this XcmsColor structure.
result_format Specifies the color format for the returned color specification.

The XcmsQueryColor function obtains the RGB value for the pixel value in the pixel member of the specified XcmsColor structure, and then converts the value to the target format as specified by the result_format argument. If the pixel is not a valid index into the specified colormap, a BadValue error results.
XcmsQueryColor can generate BadColor and BadValue errors.

To query the color of multiple colormap cells in an arbitrary format, use XcmsQueryColors.

Status XcmsQueryColors(display, colormap, colors_in_out, ncolors, result_format)

Display *display;
Colormap colormap;
XcmsColor colors_in_out[];
unsigned int ncolors;
XcmsColorFormat result_format;

display Specifies the connection to the X server.
colormap Specifies the colormap.
colors_in_out Specifies an array of XcmsColor structures, each pixel member indicating the color cell to query. The color specifications for the color cells are returned in these structures.
ncolors Specifies the number of XcmsColor structures in the color specification array.
result_format Specifies the color format for the returned color specification.

The XcmsQueryColors function obtains the RGB values for pixel values in the pixel members of XcmsColor structures, and then converts the values to the target format as specified by the result_format argument. If a pixel is not a valid index into the specified colormap, a BadValue error results. If more than one pixel is in error, the one that gets reported is arbitrary.

XcmsQueryColors can generate BadColor and BadValue errors.

6.8. Color Conversion Context Functions

This section describes functions to create, modify, and query Color Conversion Contexts. Associated with each colormap is an initial CCC transparently generated by Xlib. Therefore, when you specify a colormap as an argument to a function, you are indirectly specifying a CCC. The CCC attributes that can be modified by the X client are:
• Client White Point
• Gamut compression procedure and client data
• White point adjustment procedure and client data

The initial values for these attributes are implementation specific. The CCC attributes for subsequently created CCCs can be defined by changing the CCC attributes of the default CCC. There is a default CCC associated with each screen.

6.8.1. Getting and Setting the Color Conversion Context of a Colormap

To obtain the CCC associated with a colormap, use XcmsCCCOfColormap.

XcmsCCC XcmsCCCOfColormap(display, colormap)

Display *display;
Colormap colormap;

display Specifies the connection to the X server.
colormap Specifies the colormap.

The XcmsCCCOfColormap function returns the CCC associated with the specified colormap. Once obtained, the CCC attributes can be queried or modified. Unless the CCC associated with the specified colormap is changed with XcmsSetCCCOfColormap, this CCC is used when the specified colormap is used as an argument to color functions.
To change the CCC associated with a colormap, use XcmsSet CCCOfColormap.

XcmsCCC XcmsSet CCCOfColormap(display, colormap, ccc)
    Display *display;
    Colormap colormap;
    XcmsCCC ccc;

display Specifies the connection to the X server.

colormap Specifies the colormap.

ccc Specifies the CCC.

The XcmsSet CCCOfColormap function changes the CCC associated with the specified colormap. It returns the CCC previously associated to the colormap. If they are not used again in the application, CCCs should be freed by calling XcmsFree CCC.

6.8.2. Obtaining the Default Color Conversion Context

The default CCC attributes for subsequently created CCCs can be changed by changing the CCC attributes of the default CCC. A default CCC is associated with each screen.

To obtain the default CCC for a screen, use XcmsDefault CCC.

XcmsCCC XcmsDefault CCC(display, screen_number)
    Display *display;
    int screen_number;

display Specifies the connection to the X server.

screen_number Specifies the appropriate screen number on the host server.

The XcmsDefault CCC function returns the default CCC for the specified screen. Its visual is the default visual of the screen. Its initial gamut compression and white point adjustment procedures as well as the associated client data are implementation specific.

6.8.3. Color Conversion Context Macros

Applications should not directly modify any part of the XcmsCCC. The following lists the C language macros, their corresponding function equivalents that are for other language bindings, and what data they both can return.

DisplayOfCCC(ccc)
    XcmsCCC ccc;

Display *DisplayOfCCC(ccc)
    XcmsCCC ccc;

ccc Specifies the CCC.
Both return the display associated with the specified CCC.

VisualOfCCC(ccc)
    XcmsCCC ccc;

Visual *VisualOfCCC(ccc)
    XcmsCCC ccc;

ccc Specifies the CCC.
Both return the visual associated with the specified CCC.
Xlib - C Library

Xll, Release 5

ScreenNumberOfCCC(ccc)
    XcmsCCC ccc;

int XcmsScreenNumberOfCCC(ccc)
    XcmsCCC ccc;

ccc    Specifies the CCC.
Both return the number of the screen associated with the specified CCC.

ScreenWhitePointOfCCC(ccc)
    XcmsCCC ccc;

XcmsColor *XcmsScreenWhitePointOfCCC(ccc)
    XcmsCCC ccc;

ccc    Specifies the CCC.
Both return the white point of the screen associated with the specified CCC.

ClientWhitePointOfCCC(ccc)
    XcmsCCC ccc;

XcmsColor *XcmsClientWhitePointOfCCC(ccc)
    XcmsCCC ccc;

ccc    Specifies the CCC.
Both return the Client White Point of the specified CCC.

6.8.4. Modifying Attributes of a Color Conversion Context
To set the Client White Point in the CCC, use XcmsSetWhitePoint.

Status XcmsSetWhitePoint(ccc, color)
    XcmsCCC ccc;
    XcmsColor *color;

ccc    Specifies the CCC.
color    Specifies the new Client White Point.

The XcmsSetWhitePoint function changes the Client White Point in the specified CCC. Note
that the pixel member is ignored and that the color specification is left unchanged upon return.
The format for the new white point must be XcmsCIEXYZFormat, XcmsCIEuvYFormat,
XcmsCIExyYFormat, or XcmsUndefinedFormat. If color is NULL, this function sets the
format component of the Client White Point specification to XcmsUndefinedFormat, indicating
that the Client White Point is assumed to be the same as the Screen White Point.

To set the gamut compression procedure and corresponding client data in a specified CCC, use
XcmsSetCompressionProc.

XcmsCompressionProc XcmsSetCompressionProc(ccc, compression_proc, client_data)
    XcmsCCC ccc;
    XcmsCompressionProc compression_proc;
    XPointer client_data;

ccc    Specifies the CCC.
compression_proc    Specifies the gamut compression procedure that is to be applied when a color
lies outside the screen's color gamut. If NULL and when functions using this
CCC must convert a color specification to a device-dependent format and
encounters a color that lies outside the screen's color gamut, that function will
return XcmsFailure.

client_data Specifies client data for the gamut compression procedure or NULL.

The XcmsSetCompressionProc function first sets the gamut compression procedure and client
data in the specified CCC with the newly specified procedure and client data and then returns
the old procedure.

To set the white point adjustment procedure and corresponding client data in a specified CCC,
use XcmsSetWhiteAdjustProc.

XcmsWhiteAdjustProc XcmsSetWhiteAdjustProc(ccc, white_adjust_proc, client_data)
XcmsCCC ccc;
XcmsWhiteAdjustProc white_adjust_proc;
XPointer client_data;

ccc Specifies the CCC.

white_adjust_proc Specifies the white point adjustment procedure.

client_data Specifies client data for the white point adjustment procedure or NULL.

The XcmsSetWhiteAdjustProc function first sets the white point adjustment procedure and client
data in the specified CCC with the newly specified procedure and client data and then returns
the old procedure.

6.8.5. Creating and Freeing a Color Conversion Context
You can explicitly create a CCC within your application by calling XcmsCreateCCC. These
created CCCs can then be used by those functions that explicitly call for a CCC argument.
Old CCCs that will not be used by the application should be freed using XcmsFreeCCC.

To create a CCC, use XcmsCreateCCC.

XcmsCCC XcmsCreateCCC(display, screen_number, visual, client_white_point, compression_proc,
compression_client_data, white_adjust_proc, white_adjust_client_data)
Display *display;
int screen_number;
Visual *visual;
XcmsColor *client_white_point;
XcmsCompressionProc compression_proc;
XPointer compression_client_data;
XcmsWhiteAdjustProc white_adjust_proc;
XPointer white_adjust_client_data;

display Specifies the connection to the X server.

screen_number Specifies the appropriate screen number on the host server.

visual Specifies the visual type.

client_white_point Specifies the Client White Point. If NULL, the Client White Point is to be
assumed to be the same as the Screen White Point. Note that the pixel
member is ignored.

compression_proc Specifies the gamut compression procedure that is to be applied when a color
lies outside the screen's color gamut. If NULL and when functions using this
CCC must convert a color specification to a device-dependent format and encounters a color that lies outside the screen’s color gamut, that function will return XcmsFailure.

`compression_client_data`  
Specifies client data for use by the gamut compression procedure or NULL.

`white_adjust_proc`  
Specifies the white adjustment procedure that is to be applied when the Client White Point differs from the Screen White Point. NULL indicates that no white point adjustment is desired.

`white_adjust_client_data`  
Specifies client data for use with the white point adjustment procedure or NULL.

The XcmsCreateCCC function creates a CCC for the specified display, screen, and visual.

To free a CCC, use XcmsFreeCCC.

```c
void XcmsFreeCCC(ccc)
XcmsCCC ccc;
```

`ccc`  
Specifies the CCC.

The XcmsFreeCCC function frees the memory used for the specified CCC. Note that default CCCs and those currently associated with colormaps are ignored.

### 6.9. Converting Between Color Spaces

To convert an array of color specifications in arbitrary color formats to a single destination format, use XcmsConvertColors.

```c
Status XcmsConvertColors(ccc, colors_in_out, ncolors, target_format, compression_flags_return)
XcmsCCC ccc;
XcmsColor colors_in_out[];
unsigned int ncolors;
XcmsColorFormat target_format;
Bool compression_flags_return[];
```

`ccc`  
Specifies the CCC. If conversion is between device-independent color spaces only (for example, TekHVC to CIELuv), the CCC is necessary only to specify the Client White Point.

`colors_in_out`  
Specifies an array of color specifications. Pixel members are ignored and remain unchanged upon return.

`ncolors`  
Specifies the number of XcmsColor structures in the color specification array.

`target_format`  
Specifies the target color specification format.

`compression_flags_return`  
Specifies an array of Boolean values for returning compression status. If a non-NULL pointer is supplied, each element of the array is set to True if the corresponding color was compressed, and False otherwise. Pass NULL if the compression status is not useful.

The XcmsConvertColors function converts the color specifications in the specified array of XcmsColor structures from their current format to a single target format, using the specified CCC. When the return value is XcmsFailure, the contents of the color specification array are left unchanged.

The array may contain a mixture of color specification formats (for example, 3 CIE XYZ, 2 CIE Luv, ...). Note that when the array contains both device-independent and device-
dependent color specifications, and the target_format argument specifies a device-dependent format (for example, XcmsRGBiFormat, XcmsRGBFormat) all specifications are converted to CIE XYZ format then to the target device-dependent format.

6.10. Callback Functions

This section describes the gamut compression and white point adjustment callbacks.

The gamut compression procedure specified in the Color Conversion Context is called when an attempt to convert a color specification from XcmsCIEXYZ to a device-dependent format (typically XcmsRGBi) results in a color that lies outside the screen’s color gamut. If the gamut compression procedure requires client data, this data is passed via the gamut compression client data in the CCC.

During color specification conversion between device-independent and device-dependent color spaces, if a white point adjustment procedure is specified in the CCC, it is triggered when the Client White Point and Screen White Point differ. If required, the client data is obtained from the CCC.

6.10.1. Prototype Gamut Compression Procedure

The gamut compression callback interface must adhere to the following:

define Status (*XcmsCompressionProc)(ccc, colors_in_out, ncolors, index, compression_flags_return)

XcmsCCC ccc;
XcmsColor colors_in_out[];
unsigned int ncolors;
unsigned int index;
Bool compression_flags_return[];

ccc Specifies the CCC.

colors_in_out Specifies an array of color specifications. Pixel members are ignored and remain unchanged upon return.

ncolors Specifies the number of XcmsColor structures in the color specification array.

index Specifies the index into the array of XcmsColor structures for the encountered color specification that lies outside the Screen’s color gamut. Valid values are 0 (for the first element) to ncolors – 1.

compression_flags_return Specifies an array of Boolean values for returning compression status. If a non-NULL pointer is supplied, and a color at a given index is compressed, then True should be stored at the corresponding index in this array.

When implementing a gamut compression procedure, consider the following rules and assumptions:

- The gamut compression procedure can attempt to compress one or multiple specifications at a time.
- When called, elements 0 to index – 1 in the array of color specification array can be assumed to fall within the screen’s color gamut. In addition these color specifications are already in some device-dependent format (typically XcmsRGBi). If any modifications are made to these color specifications, they must upon return be in their initial device-dependent format.
- When called, the element in the color specification array specified by the index argument contains the color specification outside the screen’s color gamut encountered by the calling routine. In addition this color specification can be assumed to be in XcmsCIEXYZ. Upon return, this color specification must be in XcmsCIEXYZ.
- When called, elements from index to ncolors – 1 in the color specification array may or may not fall within the screen’s color gamut. In addition these color specifications can
be assumed to be in XcmsCIEXYZ. If any modifications are made to these color specifications, they must upon return be in XcmsCIEXYZ.

- The color specifications passed to the gamut compression procedure have already been adjusted to the Screen White Point. This means that at this point the color specification’s white point is the Screen White Point.
- If the gamut compression procedure uses a device-independent color space not initially accessible for use in the color management system, use XcmsAddColorSpace to insure that it is added.

6.10.2. Supplied Gamut Compression Procedures

The following equations are useful in describing gamut compression procedures.

\[ CIELab \text{ Psychometric Chroma} = \sqrt{a_{\text{star}}^2 + b_{\text{star}}^2} \]

\[ CIELab \text{ Psychometric Hue} = \tan^{-1} \left( \frac{b_{\text{star}}}{a_{\text{star}}} \right) \]

\[ CIELuv \text{ Psychometric Chroma} = \sqrt{u_{\text{star}}^2 + v_{\text{star}}^2} \]

\[ CIELuv \text{ Psychometric Hue} = \tan^{-1} \left( \frac{v_{\text{star}}}{u_{\text{star}}} \right) \]

The gamut compression callback procedures provided by Xlib are as follows.

**XcmsCIELabClipL**

Brings the encountered out of gamut color specification into the screen’s color gamut by reducing or increasing CIE metric lightness (L*) in the CIE L*a*b* color space until the color is within the gamut. If the Psychometric Chroma of the color specification is beyond maximum for the Psychometric Hue Angle, then, while maintaining the same Psychometric Hue Angle, the color will be clipped to the CIE L*a*b* coordinates of maximum Psychometric Chroma. See XcmsCIELabQueryMaxC. No client data is necessary.

**XcmsCIELabClipab**

Brings the encountered out of gamut color specification into the screen’s color gamut by reducing Psychometric Chroma, while maintaining Psychometric Hue Angle, until the color is within the gamut. No client data is necessary.

**XcmsCIELabClipLab**

Brings the encountered out of gamut color specification into the screen’s color gamut by replacing it with CIE L*a*b* coordinates that fall within the color gamut while maintaining the original Psychometric Hue Angle and whose vector to the original coordinates is the shortest attainable. No client data is necessary.

**XcmsCIELuvClipL**

Brings the encountered out of gamut color specification into the screen’s color gamut by reducing or increasing CIE metric lightness (L*) in the CIE L*u*v* color space until the color is within the gamut. If the Psychometric Chroma of the color specification is beyond maximum for the Psychometric Hue Angle, then, while maintaining the same Psychometric Hue Angle, the color will be clipped to the CIE L*u*v* coordinates of maximum Psychometric Chroma. See XcmsCIELuvQueryMaxC. No client data is necessary.

**XcmsCIELuvClipuv**

Brings the encountered out of gamut color specification into the screen’s color gamut by reducing Psychometric Chroma while maintaining Psychometric Hue Angle, until the
color is within the gamut. No client data is necessary.

*XcmsCIELuvClipLuv*

Brings the encountered out of gamut color specification into the screen’s color gamut by replacing it with CIE L*u*v* coordinates that fall within the color gamut while maintaining the original Psychometric Hue Angle and whose vector to the original coordinates is the shortest attainable. No client data is necessary.

*XcmsTekHVCClipV*

Brings the encountered out of gamut color specification into the screen’s color gamut by reducing or increasing the Value dimension in the TekHVC color space until the color is within the gamut. If Chroma of the color specification is beyond maximum for the particular Hue, then, while maintaining the same Hue, the color will be clipped to the Value and Chroma coordinates that represent maximum Chroma for that particular Hue. No client data is necessary.

*XcmsTekHVCClipC*

Brings the encountered out of gamut color specification into the screen’s color gamut by reducing the Chroma dimension in the TekHVC color space until the color is within the gamut. No client data is necessary.

*XcmsTekHVCClipVC*

Brings the encountered out of gamut color specification into the screen’s color gamut by replacing it with TekHVC coordinates that fall within the color gamut while maintaining the original Hue and whose vector to the original coordinates is the shortest attainable. No client data is necessary.

6.10.3. Prototype White Point Adjustment Procedure

The white point adjustment procedure interface must adhere to the following:

```c
typedef Status (*XcmsWhiteAdjustProc)(ccc, initial_white_point, target_white_point, target_format, colors_in_out, ncolors, compression_flags_return)

XcmsCCC ccc;
XcmsColor *initial_white_point;
XcmsColor *target_white_point;
XcmsColorFormat target_format;
XcmsColor colors_in_out[];
unsigned int ncolors;
Bool compression_flags_return[];
```

*ccc* Specifies the CCC.

*initial_white_point* Specifies the initial white point.

*target_white_point* Specifies the target white point.

*target_format* Specifies the target color specification format.

*colors_in_out* Specifies an array of color specifications. Pixel members are ignored and remain unchanged upon return.

*ncolors* Specifies the number of XcmsColor structures in the color specification array.

*compression_flags_return* Specifies an array of Boolean values for returning compression status. If a non-NULL pointer is supplied, and a color at a given index is compressed, then True should be stored at the corresponding index in this array.
6.10.4. Supplied White Point Adjustment Procedures

White point adjustment procedures provided by Xlib are as follows.

XcmsCIELabWhiteShiftColors

Uses the CIE L*a*b* color space for adjusting the chromatic character of colors to compensate for the chromatic differences between the source and destination white points. This procedure simply converts the color specifications to XcmsCIELab using the source white point and then converts to the target specification format using the destinations white point. No client data is necessary.

XcmsCIELuvWhiteShiftColors

Uses the CIE L*u*v* color space for adjusting the chromatic character of colors to compensate for the chromatic differences between the source and destination white points. This procedure simply converts the color specifications to XcmsCIELuv using the source white point and then converts to the target specification format using the destinations white point. No client data is necessary.

XcmsTekHVCWhiteShiftColors

Uses the TekHVC color space for adjusting the chromatic character of colors to compensate for the chromatic differences between the source and destination white points. This procedure simply converts the color specifications to XcmsTekHVC using the source white point and then converts to the target specification format using the destinations white point. An advantage of this procedure over those previously described is an attempt to minimize hue shift. No client data is necessary.

From an implementation point of view, these white point adjustment procedures convert the color specifications to a device-independent but white-point-dependent color space (for example, CIE L*u*v*, CIE L*a*b*, TekHVC) using one white point and then converting those specifications to the target color space using another white point. In other words, the specification goes in the color space with one white point but comes out with another white point, resulting in a chromatic shift based on the chromatic displacement between the initial white point and target white point. The CIE color spaces that are assumed to be white-point-independent are CIE u’v’Y, CIE XYZ, and CIE xyY. When developing a custom white point adjustment procedure that uses a device-independent color space not initially accessible for use in the color management system, use XcmsAddColorSpace to insure that it is added.

As an example, if a white point adjustment procedure is specified by the Color Conversion Context and if the Client White Point and Screen White Point differ, the XcmsAllocColor function will use the white point adjustment procedure twice:
- Once to convert to XcmsRGB
- A second time to convert from XcmsRGB

For example, assume the specification is in XcmsCIEuvY and the adjustment procedure is XcmsCIELuvWhiteShiftColors. During conversion to XcmsRGB, the call to XcmsAllocColor results in the following series of color specification conversions:
- From XcmsCIEuvY to XcmsCIELuv using the Client White Point
- From XcmsCIELuv to XcmsCIEuvY using the Screen White Point
- From XcmsCIEuvY to XcmsCIEXYZ (CIE u’v’Y and XYZ are white-point-independent color spaces)
- From XcmsCIEXYZ to XcmsRGBi
- Finally from XcmsRGBi to XcmsRGB

The resulting RGB specification is passed to XAllocColor and the RGB specification returned by XAllocColor is converted back to XcmsCIEuvY by reversing the color conversion sequence.
6.11. Gamut Querying Functions
This section describes the gamut querying functions that are provided by Xlib. These functions allow the client to query the boundary of the screen’s color gamut in terms of the CIE L*a*b*, CIE L*u*v*, and TekHVC color spaces. Functions are also provided that allow you to query the color specification of:

- White (full intensity red, green, and blue)
- Red (full intensity red while green and blue are zero)
- Green (full intensity green while red and blue are zero)
- Blue (full intensity blue while red and green are zero)
- Black (zero intensity red, green, and blue)

The white point associated with color specifications passed to and returned from these gamut querying functions are assumed to be the Screen White Point. This is a reasonable assumption, since the client is trying to query the screen’s color gamut.

Note that the following naming convention is used for the "Max" and "Min" functions:

```
Xcms<color_space>QueryMax<dimensions>
```

```
Xcms<color_space>QueryMin<dimensions>
```

Note that the `<dimensions>` consists of letter or letters that identify the dimension or dimensions of the color space that are not fixed. For example, `XcmsTekHVCQueryMaxC` is given a fixed Hue and Value for which maximum Chroma is found.

6.11.1. Red, Green, and Blue Queries
To obtain the color specification for black (zero intensity red, green, and blue), use `XcmsQueryBlack`.

```
Status XcmsQueryBlack(ccc, targetFormat, colorReturn)
    XcmsCCC ccc;
    XcmsColorFormat targetFormat,
    XcmsColor *colorReturn;
    ccc Specifies the CCC. Note that the CCC's Client White Point and White Point Adjustment procedures are ignored.
    targetFormat Specifies the target color specification format.
    colorReturn Returns the color specification in the specified target format for zero intensity red, green, and blue. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.
```

The `XcmsQueryBlack` function returns the color specification in the specified target format for zero intensity red, green, and blue.

To obtain the color specification for blue (full intensity blue while red and green are zero), use `XcmsQueryBlue`.

```
Status XcmsQueryBlue(ccc, targetFormat, colorReturn)
    XcmsCCC ccc;
    XcmsColorFormat targetFormat,
    XcmsColor *colorReturn;
    ccc Specifies the CCC. Note that the CCC's Client White Point and White Point Adjustment procedures are ignored.
    targetFormat Specifies the target color specification format.
```

88
color_return Returns the color specification in the specified target format for full intensity blue while red and green are zero. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsQueryBlue function returns the color specification in the specified target format for full intensity blue while red and green are zero.

To obtain the color specification for green (full intensity green while red and blue are zero), use XcmsQueryGreen.

Status XcmsQueryGreen(ccc, target_format, color_return)
XcmsCCC ccc;
XcmsColorFormat target_format;
XcmsColor *color_return;

ccc Specifies the CCC. Note that the CCC’s Client White Point and White Point Adjustment procedures are ignored.
target_format Specifies the target color specification format.
color_return Returns the color specification in the specified target format for full intensity green while red and blue are zero. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsQueryGreen function returns the color specification in the specified target format for full intensity green while red and blue are zero.

To obtain the color specification for red (full intensity red while green and blue are zero), use XcmsQueryRed.

Status XcmsQueryRed(ccc, target_format, color_return)
XcmsCCC ccc;
XcmsColorFormat target_format;
XcmsColor *color_return;

ccc Specifies the CCC. Note that the CCC’s Client White Point and White Point Adjustment procedures are ignored.
target_format Specifies the target color specification format.
color_return Returns the color specification in the specified target format for full intensity red while green and blue are zero. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsQueryRed function returns the color specification in the specified target format for full intensity red while green and blue are zero.

To obtain the color specification for white (full intensity red, green, and blue), use XcmsQueryWhite.

Status XcmsQueryWhite(ccc, target_format, color_return)
XcmsCCC ccc;
XcmsColorFormat target_format;
XcmsColor *color_return;

ccc Specifies the CCC. Note that the CCC’s Client White Point and White Point Adjustment procedures are ignored.
target_format Specifies the target color specification format.
color_return Returns the color specification in the specified target format for full intensity red, green, and blue. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsQueryWhite function returns the color specification in the specified target format for full intensity red, green, and blue.

6.11.2. CIELab Queries

To obtain the CIE L*a*b* coordinates of maximum Psychometric Chroma for a given Psychometric Hue Angle and CIE metric lightness (L*), use XcmsCIELabQueryMaxC.

\[
\text{CIELab Psychometric Chroma} = \sqrt{a_\text{star}^2 + b_\text{star}^2}
\]

\[
\text{CIELab Psychometric Hue} = \tan^{-1} \left( \frac{b_\text{star}}{a_\text{star}} \right)
\]

Status XcmsCIELabQueryMaxC(ccc, hue_angle, L_star, color_return)

XcmsCCC ccc;
XcmsFloat hue_angle;
XcmsFloat L_star;
XcmsColor * color_return;

ccc Specifies the CCC. Note that the CCC's Client White Point and White Point Adjustment procedures are ignored.

hue_angle Specifies the hue angle in degrees at which to find maximum chroma.

L_star Specifies the lightness (L*) at which to find maximum chroma.

color_return Returns the CIE L*a*b* coordinates of maximum chroma displayable by the screen for the given hue angle and lightness. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsCIELabQueryMaxC function, given a hue angle and lightness, finds the point of maximum chroma displayable by the screen. It returns this point in CIE L*a*b* coordinates.

To obtain the CIE L*a*b* coordinates of maximum CIE metric lightness (L*) for a given Psychometric Hue Angle and Psychometric Chroma, use XcmsCIELabQueryMaxL.

Status XcmsCIELabQueryMaxL(ccc, hue_angle, chroma, color_return)

XcmsCCC ccc;
XcmsFloat hue_angle;
XcmsFloat chroma;
XcmsColor * color_return;

ccc Specifies the CCC. Note that the CCC's Client White Point and White Point Adjustment procedures are ignored.

hue_angle Specifies the hue angle in degrees at which to find maximum lightness.

chroma Specifies the chroma at which to find maximum lightness.

color_return Returns the CIE L*a*b* coordinates of maximum lightness displayable by the screen for the given hue angle and chroma. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsCIELabQueryMaxL function, given a hue angle and chroma, finds the point in CIE L*a*b* color space of maximum lightness (L*) displayable by the screen. It returns this
point in CIE L*a*b* coordinates. An XcmsFailure return value usually indicates that the
given chroma is beyond maximum for the given hue angle.

To obtain the CIE L*a*b* coordinates of maximum Psychometric Chroma for a given
Psychometric Hue Angle, use XcmsCIELabQueryMaxLC.

Status XcmsCIELabQueryMaxLC(ccc, hue_angle, color_return)

XcmsCCC ccc;
XcmsFloat hue_angle;
XcmsColor *color_return;

ccc Specifies the CCC. Note that the CCC’s Client White Point and White Point
Adjustment procedures are ignored.
hue_angle Specifies the hue angle in degrees at which to find maximum chroma.
color_return Returns the CIE L*a*b* coordinates of maximum chroma displayable by the
screen for the given hue angle. The white point associated with the returned
color specification is the Screen White Point. The value returned in the pixel
member is undefined.

The XcmsCIELabQueryMaxLC function, given a hue angle, finds the point of maximum
chroma displayable by the screen. It returns this point in CIE L*a*b* coordinates.

To obtain the CIE L*a*b* coordinates of minimum CIE metric lightness (L*) for a given
Psychometric Hue Angle and Psychometric Chroma, use XcmsCIELabQueryMinL.

Status XcmsCIELabQueryMinL(ccc, hue_angle, chroma, color_return)

XcmsCCC ccc;
XcmsFloat hue_angle;
XcmsFloat chroma;
XcmsColor *color_return;

ccc Specifies the CCC. Note that the CCC’s Client White Point and White Point
Adjustment procedures are ignored.
hue_angle Specifies the hue angle in degrees at which to find minimum lightness.
chroma Specifies the chroma at which to find minimum lightness.
color_return Returns the CIE L*a*b* coordinates of minimum lightness displayable by the
screen for the given hue angle and chroma. The white point associated with the returned
color specification is the Screen White Point. The value returned
in the pixel member is undefined.

The XcmsCIELabQueryMinL function, given a hue angle and chroma, finds the point of
minimum lightness (L*) displayable by the screen. It returns this point in CIE L*a*b* coordinates. An XcmsFailure return value usually indicates that the given chroma is beyond max-
imum for the given hue angle.

6.11.3. CIELuv Queries

To obtain the CIE L*u*v* coordinates of maximum Psychometric Chroma for a given
Psychometric Hue Angle and CIE metric lightness (L*), use XcmsCIELuvQueryMaxC.

\[
\text{CIELuv Psychometric Chroma} = \sqrt{u_{\text{star}}^2 + v_{\text{star}}^2}
\]

\[
\text{CIELuv Psychometric Hue} = \tan^{-1} \left[ \frac{v_{\text{star}}}{u_{\text{star}}} \right]
\]
Status XcmsCIELuvQueryMaxC(ccc, hue_angle, L_star, color_return)
XcmsCCC ccc;
XcmsFloat hue_angle;
XcmsFloat L_star;
XcmsColor *color_return;

ccc Specifies the CCC. Note that the CCC's Client White Point and White Point Adjustment procedures are ignored.
hue_angle Specifies the hue angle in degrees at which to find maximum chroma.
L_star Specifies the lightness (L*) at which to find maximum chroma.
color_return Returns the CIE L*u*v* coordinates of maximum chroma displayable by the screen for the given hue angle and lightness. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsCIELuvQueryMaxC function, given a hue angle and lightness, finds the point of maximum chroma displayable by the screen. Note that it returns this point in CIE L*u*v* coordinates.

To obtain the CIE L*u*v* coordinates of maximum CIE metric lightness (L*) for a given Psychometric Hue Angle and Psychometric Chroma, use XcmsCIELuvQueryMaxL.

Status XcmsCIELuvQueryMaxL(ccc, hue_angle, chroma, color_return)
XcmsCCC ccc;
XcmsFloat hue_angle;
XcmsFloat chroma;
XcmsColor *color_return;

ccc Specifies the CCC. Note that the CCC's Client White Point and White Point Adjustment procedures are ignored.
hue_angle Specifies the hue angle in degrees at which to find maximum lightness.
L_star Specifies the lightness (L*) at which to find maximum lightness.
color_return Returns the CIE L*u*v* coordinates of maximum lightness displayable by the screen for the given hue angle and chroma. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsCIELuvQueryMaxL function, given a hue angle and chroma, finds the point in CIE L*u*v* color space of maximum lightness (L*) displayable by the screen. Note that it returns this point in CIE L*u*v* coordinates. An XcmsFailure return value usually indicates that the given chroma is beyond maximum for the given hue angle.

To obtain the CIE L*u*v* coordinates of maximum Psychometric Chroma for a given Psychometric Hue Angle, use XcmsCIELuvQueryMaxLC.

Status XcmsCIELuvQueryMaxLC(ccc, hue_angle, color_return)
XcmsCCC ccc;
XcmsFloat hue_angle;
XcmsColor *color_return;

ccc Specifies the CCC. Note that the CCC's Client White Point and White Point Adjustment procedures are ignored.
hue_angle Specifies the hue angle in degrees at which to find maximum chroma.
color_return Returns the CIE L*u*v* coordinates of maximum chroma displayable by the screen for the given hue angle. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel
member is undefined.

The XcmsCIELuvQueryMaxLC function, given a hue angle, finds the point of maximum chroma displayable by the screen. Note that it returns this point in CIE L*u*v* coordinates.

To obtain the CIE L*u*v* coordinates of minimum CIE metric lightness (L*) for a given Psychometric Hue Angle and Psychometric Chroma, use XcmsCIELuvQueryMinL.

Status XcmsCIELuvQueryMinL(ccc, hue_angle, chroma, color_return)

XcmsCCC ccc;
XcmsFloat hue_angle;
XcmsFloat chroma;
XcmsColor *color_return;

cc c Specifies the CCC. Note that the CCC’s Client White Point and White Point Adjustment procedures are ignored.

hue_angle Specifies the hue angle in degrees at which to find minimum lightness.

chroma Specifies the chroma at which to find minimum lightness.

color_return Returns the CIE L*u*v* coordinates of minimum lightness displayable by the screen for the given hue angle and chroma. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsCIELuvQueryMinL function, given a hue angle and chroma, finds the point of minimum lightness (L*) displayable by the screen. Note that it returns this point in CIE L*u*v* coordinates. An XcmsFailure return value usually indicates that the given chroma is beyond maximum for the given hue angle.

6.11.4. TekHVC Queries

To obtain the maximum Chroma for a given Hue and Value, use XcmsTekHVCQueryMaxC.

Status XcmsTekHVCQueryMaxC(ccc, hue, value, color_return)

XcmsCCC ccc;
XcmsFloat hue;
XcmsFloat value;
XcmsColor *color_return;

cc c Specifies the CCC. Note that the CCC’s Client White Point and White Point Adjustment procedures are ignored.

hue Specifies the Hue in which to find the maximum Chroma.

value Specifies the Value in which to find the maximum Chroma.

color_return Returns the maximum Chroma along with the actual Hue and Value at which the maximum Chroma was found. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsTekHVCQueryMaxC function, given a Hue and Value, determines the maximum Chroma in TekHVC color space displayable by the screen. Note that it returns the maximum Chroma along with the actual Hue and Value at which the maximum Chroma was found.

To obtain the maximum Value for a given Hue and Chroma, use XcmsTekHVCQueryMaxV
Status XcmsTekHVCQueryMaxV(ccc, hue, chroma, color_return)
    XcmsCCC ccc;
    XcmsFloat hue;
    XcmsFloat chroma;
    XcmsColor *color_return;

ccc  Specifies the CCC. Note that the CCC's Client White Point and White Point
     Adjustment procedures are ignored.

hue  Specifies the Hue in which to find the maximum Value.

croma Specifies the chroma at which to find maximum Value.

color_return  Returns the maximum Value along with the Hue and Chroma at which the
               maximum Value was found. The white point associated with the returned
               color specification is the Screen White Point. The value returned in the pixel
               member is undefined.

The XcmsTekHVCQueryMaxV function, given a Hue and Chroma, determines the maximum
Value in TekHVC color space displayable by the screen. Note that it returns the maximum
Value and the actual Hue and Chroma at which the maximum Value was found.

To obtain the maximum Chroma and Value at which it is reached for a specified Hue, use
XcmsTekHVCQueryMaxVC.

Status XcmsTekHVCQueryMaxVC(ccc, hue, color_return)
    XcmsCCC ccc;
    XcmsFloat hue;
    XcmsColor *color_return;

ccc  Specifies the CCC. Note that the CCC's Client White Point and White Point
     Adjustment procedures are ignored.

hue  Specifies the Hue in which to find the maximum Chroma.

color_return  Returns the color specification in XcmsTekHVC for the maximum Chroma, the
               Value at which that maximum Chroma is reached and actual Hue at which the
               maximum Chroma was found. The white point associated with the returned
               color specification is the Screen White Point. The value returned in the pixel
               member is undefined.

The XcmsTekHVCQueryMaxVC function, given a Hue, determines the maximum Chroma in
TekHVC color space displayable by the screen and the Value at which that maximum Chroma
is reached. Note that it returns the maximum Chroma, the Value at which that maximum
Chroma is reached, and the actual Hue for which the maximum Chroma was found.

To obtain a specified number of TekHVC specifications such that they contain a maximum
Values for a specified Hue, and the Chroma at which the maximum Values are reached, use
XcmsTekHVCQueryMaxVSamples.

Status XcmsTekHVCQueryMaxVSamples(ccc, hue, colors_return, nsamples)
    XcmsCCC ccc;
    XcmsFloat hue;
    XcmsColor colors_return[];
    unsigned int nsamples;

ccc  Specifies the CCC. Note that the CCC's Client White Point and White Point
     Adjustment procedures are ignored.

hue  Specifies the Hue for maximum Chroma/Value samples.

nsamples  Specifies the number of samples.
colors_in_out Returns nsamples of color specifications in XcmsTekHVC such that the Chroma is the maximum attainable for the Value and Hue. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsTekHVCQueryMaxVSamples returns nsamples of maximum Value, Chroma at which that maximum Value is reached, and the actual Hue for which the maximum Chroma was found. These sample points may then be used to plot the maximum Value/Chroma boundary of the screen's color gamut for the specified Hue in TekHVC color space.

To obtain the minimum Value for a given Hue and Chroma, use XcmsTekHVCQueryMinV.

Status XcmsTekHVCQueryMinV(ccc, hue, chroma, color_return)

XcmsCCC ccc;
XcmsFloat hue;
XcmsFloat chroma;
XcmsColor *color_return;

ccc Specifies the CCC. Note that the CCC's Client White Point and White Point Adjustment procedures are ignored.

hue Specifies the Hue in which to find the minimum Value.

value Specifies the Value in which to find the minimum Value.

color_return Returns the minimum Value and the actual Hue and Chroma at which the minimum Value was found. The white point associated with the returned color specification is the Screen White Point. The value returned in the pixel member is undefined.

The XcmsTekHVCQueryMinV function, given a Hue and Chroma, determines the minimum Value in TekHVC color space displayable by the screen. Note that it returns the minimum Value and the actual Hue and Chroma at which the minimum Value was found.

6.12. Color Management Extensions

The Xlib color management facilities can be extended in two ways:

• Device-Independent Color Spaces
  Device-independent color spaces that are derivable to CIE XYZ space can be added using the XcmsAddColorSpace function.

• Color Characterization Function Set
  A Color Characterization Function Set consists of device-dependent color spaces and their functions that convert between these color spaces and the CIE XYZ color space, bundled together for a specific class of output devices. A function set can be added using the XcmsAddFunctionSet function.

6.12.1. Color Spaces

The CIE XYZ color space serves as the "hub" for all conversions between device-independent and device-dependent color spaces. Therefore, associated with each color space is the knowledge to convert an XcmsColor structure to and from CIE XYZ format. For example, conversion from CIE L*u*v* to RGB requires the knowledge to convert from CIE L*u*v* to CIE XYZ and from CIE XYZ to RGB. This knowledge is stored as an array of functions that when applied in series will convert the XcmsColor structure to or from CIE XYZ format. This color specification conversion mechanism facilitates the addition of color spaces.

Of course, when converting between only device-independent color spaces or only device-dependent color spaces, short cuts are taken whenever possible. For example, conversion from TekHVC to CIE L*u*v* is performed by intermediate conversion to CIE u*v*Y and then to CIE L*u*v*, thus bypassing conversion between CIE u*v*Y and CIE XYZ.
6.12.2. Adding Device-Independent Color Spaces
To add a device-independent color space, use XcmsAddColorSpace.

Status XcmsAddColorSpace (color_space)
   XcmsColorSpace *color_space;

   color_space: Specifies the device-independent color space to add.

The XcmsAddColorSpace function makes a device-independent color space (actually an
XcmsColorSpace structure) accessible by the color management system. Because format
values for unregistered color spaces are assigned at run-time, they should be treated as private
to the client. If references to an unregistered color space must be made outside the client (for
example, storing color specifications in a file using the unregistered color space), then reference
should be made by color space prefix (see XcmsFormatOfPrefix and XcmsPrefixOfFormat).

If the XcmsColorSpace structure is already accessible in the color management system,
XcmsAddColorSpace returns XcmsSuccess.

Note that added XcmsColorSpaces must be retained for reference by Xlib.

6.12.3. Querying Color Space Format and Prefix
To obtain the format associated with the color space associated with a specified color string
prefix, use XcmsFormatOfPrefix.

XcmsColorFormat XcmsFormatOfPrefix (prefix)
   char *prefix;

   prefix: Specifies the string that contains the color space prefix.

The XcmsFormatOfPrefix function returns format for the specified color space prefix (for
example, "CIEXYZ"). Note that the prefix is case-insensitive. If the color space is not acces¬
sible in the color management system, XcmsFormatOfPrefix returns
XcmsUndefinedFormat.

To obtain the color string prefix associated with the color space specified by a color format,
use XcmsPrefixOfFormat.

char *XcmsPrefixOfFormat (format)
   XcmsColorFormat format;

   format: Specifies the color specification format.

The XcmsPrefixOfFormat function returns the string prefix associated with the color
specification encoding specified by format. Otherwise, if none is found, it returns NULL.
Note that the returned string must be treated as read-only.

6.12.4. Creating Additional Color Spaces
Color space specific information necessary for color space conversion and color string parsing
is stored in an XcmsColorSpace structure. Therefore, a new structure containing this infor¬
mation is required for each additional color space. In the case of device-independent color
spaces, a handle to this new structure (that is, by means of a global variable) is usually made
accessible to the client program for use with the XcmsAddColorSpace function.

If a new XcmsColorSpace structure specifies a color space not registered with the X Consor¬
tium, because format values for unregistered color spaces are assigned at run-time they should
be treated as private to the client. If references to an unregistered color space must be made
outside the client (for example, storing color specifications in a file using the unregistered color
space), then reference should be made by color space prefix (see XcmsFormatOfPrefix and
XcmsPrefixOfFormat).

typedef (*XcmsConversionProc());
typedef XcmsConversionProc *XcmsFuncListPtr;
typedef struct _XcmsColorSpace {
    char *prefix;
    XcmsColorFormat format;
    XcmsParseStringProc parseString;
    XcmsFuncListPtr to_CIEXYZ;
    XcmsFuncListPtr from_CIEXYZ;
    int inverse_flag;
} XcmsColorSpace;

The prefix member specifies the prefix that indicates a color string is in this color space’s string format. For example, “ciexyz” or “CIEXYZ” for CIE XYZ, and “rgb” or “RGB” for RGB. Note that the prefix is case insensitive. The format member specifies the color specification format. Formats for unregistered color spaces are assigned at run-time. The parseString member contains a pointer to the function that can parse a color string into an XcmsColor structure. This function returns an integer (int): non-zero if it succeeded and zero otherwise. The to_CIEXYZ and from_CIEXYZ members contain pointers, each to a NULL terminated list of function pointers. When the list of functions are executed in series, it will convert the color specified in an XcmsColor structure from/to the current color space format to/from the CIE XYZ format. Each function returns an integer (int): non-zero if it succeeded and zero otherwise. Note that the white point to be associated with the colors is specified explicitly, even though white points can be found in the Color Conversion Context. The inverse_flag member, if non-zero, specifies that for each function listed in to_CIEXYZ, its inverse function can be found in from_CIEXYZ such that:

Given: \( n = \text{number of functions in each list} \)

foreach \( i \), such that \( 0 <= i < n \)

   from_CIEXYZ[\( n - i - 1 \)] is the inverse of to_CIEXYZ[\( i \)].

This allows Xlib to use the shortest conversion path, thus, bypassing CIE XYZ if possible (for example, TekHVC to CIE L*U*V*).

6.12.5. Parse String Callback

The callback in the XcmsColorSpace structure for parsing a color string for the particular color space must adhere to the following software interface specification:

typedef int (*XcmsParseStringProc)(color_string, color_return)
    char *color_string;
    XcmsColor *color_return; /* color to compress */

color_string Specifies the color string to parse.

color_return Returns the color specification in the color space’s format.


Callback functions in the XcmsColorSpace structure for converting a color specification between device-independent spaces must adhere to the following software interface specification:

Status ConversionProc(ccc, white_point, colors_in_out, ncolors)
    XcmsCCC ccc;
    XcmsColor *white_point;
    XcmsColor *colors_in_out;
    unsigned int ncolors;
ccc Specifies the CCC.

white_point Specifies the white point associated with color specifications. Note that the pixel member is ignored and that the color specification is left unchanged upon return.

colors_in_out Specifies an array of color specifications. Pixel members are ignored and remain unchanged upon return.

ncolors Specifies the number of XcmsColor structures in the color specification array.

Callback functions in the XcmsColorSpace structure for converting a color specification to or from a device-dependent space must adhere to the following software interface specification:

Status ConversionProc(ccc, colors_in_out, ncolors, compression_flags_return)

XcmsCCC ccc;
XcmsColor *colors_in_out;
unsigned int ncolors;
Bool compression_flags_return[];

cccc Specifies the CCC.

colors_in_out Specifies an array of color specifications. Pixel members are ignored and remain unchanged upon return.

ncolors Specifies the number of XcmsColor structures in the color specification array.

compression_flags_return Specifies an array of Boolean values for returning compression status. If a non-NULL pointer is supplied, and a color at a given index is compressed, then True should be stored at the corresponding index in this array.

Conversion functions are available globally for use by other color spaces. The conversion functions provided by Xlib are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Converts</th>
</tr>
</thead>
<tbody>
<tr>
<td>XcmsCIELabToCIEXYZ</td>
<td>From XcmsCIELabFormat to XcmsCIEXYZFormat.</td>
</tr>
<tr>
<td>XcmsCIELuvToCIEuvY</td>
<td>From XcmsCIELuvFormat to XcmsCIEuvYFormat.</td>
</tr>
<tr>
<td>XcmsCIEXYZToCIELab</td>
<td>From XcmsCIEXYZFormat to XcmsCIELabFormat.</td>
</tr>
<tr>
<td>XcmsCIEXYZToCIEuvY</td>
<td>From XcmsCIEXYZFormat to XcmsCIEuvYFormat.</td>
</tr>
<tr>
<td>XcmsCIEXYZToCIExyY</td>
<td>From XcmsCIEXYZFormat to XcmsCIExyYFormat.</td>
</tr>
<tr>
<td>XcmsCIEXYZToRgbI</td>
<td>From XcmsCIEXYZFormat to XcmsRgbIFormat.</td>
</tr>
<tr>
<td>XcmsCIEuvYToCIEXYZ</td>
<td>From XcmsCIEuvYFormat to XcmsCIEXYZFormat.</td>
</tr>
<tr>
<td>XcmsCIEuvYToCIELab</td>
<td>From XcmsCIEuvYFormat to XcmsCIEXYZFormat.</td>
</tr>
<tr>
<td>XcmsCIEuvYToTekHVC</td>
<td>From XcmsCIEuvYFormat to XcmsTekHVCFormat.</td>
</tr>
<tr>
<td>XcmsCIExyYToCIEXYZ</td>
<td>From XcmsCIExyYFormat to XcmsCIEXYZFormat.</td>
</tr>
<tr>
<td>XcmsRgbToRgbI</td>
<td>From XcmsRgbFormat to XcmsRgbIFormat.</td>
</tr>
<tr>
<td>XcmsRgbIToCIEXYZ</td>
<td>From XcmsRgbIFormat to XcmsCIEXYZFormat.</td>
</tr>
<tr>
<td>XcmsRgbIToRgb</td>
<td>From XcmsRgbIFormat to XcmsRgbFormat.</td>
</tr>
<tr>
<td>XcmsTekHVCToCIEuvY</td>
<td>From XcmsTekHVCFormat to XcmsCIEuvYFormat.</td>
</tr>
</tbody>
</table>
6.12.7. Function Sets
Functions to convert between device-dependent color spaces and CIE XYZ may differ for different classes of output devices (for example, color versus gray monitors). Therefore, the notion of a Color Characterization Function Set (hereafter referred to as a Function Set) has been developed. A function set consists of device-dependent color spaces and the functions that convert color specifications between these device-dependent color spaces and the CIE XYZ color space appropriate for a particular class of output devices. The function set also contains a function that reads color characterization data off root window properties. It is this characterization data that will differ between devices within a class of output devices. For details about color characterization data is stored in root window properties, see the section on Device Color Characterization in the *Inter-Client Communication Conventions Manual*. The LINEAR_RGB Function Set is provided by Xlib and will support most color monitors. Function sets may require data that differs from those needed for the LINEAR_RGB Function Set. In that case, its corresponding data may be stored on different root window properties.

6.12.8. Adding Function Sets
To add a Color Characterization Function Set, use XcmsAddFunctionSet.

*Status* XcmsAddFunctionSet(function_set)

```
XcmsFunctionSet *function_set;
```

*function_set* Specifies the Color Characterization Function Set to add.

The XcmsAddFunctionSet adds a Color Characterization Function Set to the color management system. If the Function Set uses device-dependent XcmsColorSpace structures not accessible in the color management system, XcmsAddFunctionSet adds them. If an added XcmsColorSpace structure is for a device-dependent color space not registered with the X Consortium, because format values for unregistered color spaces are assigned at run-time they should be treated as private to the client. If references to an unregistered color space must be made outside the client (for example, storing color specifications in a file using the unregistered color space), then reference should be made by color space prefix (see XcmsFormatQfPrefix and XcmsPrefixOfFormat).

Additional function sets should be added before any calls to other Xlib routines are made. If not, the XcmsPerScrInfo member of a previously created XcmsCCC does not have the opportunity to initialize with the added Function Set.

6.12.9. Creating Additional Function Sets
Creation of additional Color Characterization Function Sets should be required only when an output device does not conform to existing function sets or when additional device-dependent color spaces are necessary. A function set consists primarily of a collection of device-dependent XcmsColorSpace structures and a means to read and store a screen’s color characterization data. This data is stored in an XcmsFunctionSet structure. A handle to this structure (that is, by means of global variable) is usually made accessible to the client program for use with XcmsAddFunctionSet.

If a Function Set uses new device-dependent XcmsColorSpace structures, they will be transparently processed into the color management system. Function Sets can share an XcmsColorSpace structure for a device-dependent color space. In addition, multiple XcmsColorSpace structures are allowed for a device-dependent color space; however, a Function Set can reference only one of them. These XcmsColorSpace structures will differ in the functions to convert to and from CIE XYZ, thus tailored for the specific Function Set.

```
typedef struct _XcmsFunctionSet {
    XcmsColorSpace **DDColorSpaces;
    XcmsScreenInitProc screenInitProc;
    XcmsScreenFreeProc screenFreeProc;
} XcmsFunctionSet;
```
The DDColorSpaces member is a pointer to a NULL terminated list of pointers to XcmsColorSpace structures for the device-dependent color spaces that are supported by the Function Set. The screenInitProc member is set to the callback procedure (see following interface specification) that initializes the XcmsPerScrnInfo structure for a particular screen.

The screen initialization callback must adhere to the following software interface specification:

typedef Status (*XcmsScreenInitProc)(Display, screen_number, screen_info)
    Display *display;
    int screen_number;
    XcmsPerScrnInfo *screen_info;

display Specifies the connection to the X server.
screen_number Specifies the appropriate screen number on the host server.
screen_info Specifies the XcmsPerScrnInfo structure, which contains the per screen information.

The screen initialization callback in the XcmsFunctionSet structure fetches the Color Characterization Data (device profile) for the specified screen, typically off properties on the screen’s root window; then it initializes the specified XcmsPerScrnInfo structure. If successful, the procedure fills in the XcmsPerScrnInfo structure as follows:

• It sets the screenData member to the address of the created device profile data structure (contents known only by the function set).
• It next sets the screenWhitePoint member.
• It next sets the functionSet member to the address of the XcmsFunctionSet structure.
• It then sets the state member to XcmsInitSuccess and finally returns XcmsSuccess.

If unsuccessful, the procedure sets the state member to XcmsInitFailure and returns XcmsFailure.

The XcmsPerScrnInfo structure contains:

typedef struct XcmsPerScrnInfo {
    XcmsColor screenWhitePoint;
    XPointer functionSet;
    XPointer screenData;
    unsigned char state;
    char pad[3];
} XcmsPerScrnInfo;

The screenWhitePoint member specifies the white point inherent to the screen. The functionSet member specifies the appropriate Function Set. The screenData member specifies the device profile. The state member is set to one of the following:

• XcmsInitNone indicates initialization has not been previously attempted.
• XcmsInitFailure indicates initialization has been previously attempted but failed.
• XcmsInitSuccess indicates initialization has been previously attempted and succeeded.

The screen free callback must adhere to the following software interface specification:

typedef void (*XcmsScreenFreeProc)(screenData)
    XPointer screenData;

screenData Specifies the data to be freed.

This function is called to free the screenData stored in an XcmsPerScrnInfo structure.
Chapter 7

Graphics Context Functions

A number of resources are used when performing graphics operations in X. Most information about performing graphics (for example, foreground color, background color, line style, and so on) are stored in resources called graphics contexts (GC). Most graphics operations (see chapter 8) take a GC as an argument. Although in theory the X protocol permits sharing of GCs between applications, it is expected that applications will use their own GCs when performing operations. Sharing of GCs is highly discouraged because the library may cache GC state.

Graphics operations can be performed to either windows or pixmaps, which collectively are called drawables. Each drawable exists on a single screen. A GC is created for a specific screen and drawable depth, and can only be used with drawables of matching screen and depth.

7.1. Manipulating Graphics Context/State

Most attributes of graphics operations are stored in Graphic Contexts (GCs). These include line width, line style, plane mask, foreground, background, tile, stipple, clipping region, end style, join style, and so on. Graphics operations (for example, drawing lines) use these values to determine the actual drawing operation. Extensions to X may add additional components to GCs. The contents of a GC are private to Xlib.

Xlib implements a write-back cache for all elements of a GC that are not resource IDs to allow Xlib to implement the transparent coalescing of changes to GCs. For example, a call to XSetForeground of a GC followed by a call to XSetLineAttributes results in only a single-change GC protocol request to the server. GCs are neither expected nor encouraged to be shared between client applications, so this write-back caching should present no problems. Applications cannot share GCs without external synchronization. Therefore, sharing GCs between applications is highly discouraged.

To set an attribute of a GC, set the appropriate member of the XGCValues structure and OR in the corresponding value bitmask in your subsequent calls to XCreateGC. The symbols for the value mask bits and the XGCValues structure are:

/* GC attribute value mask bits */

#define GCFunction (1L<<0)
#define GCPianeMask (1L<<1)
#define GCForeground (1L<<2)
#define GCBackground (1L<<3)
#define GCLLineWidth (1L<<4)
#define GCLineStyle (1L<<5)
#define GCCapStyle (1L<<6)
#define GCJoinStyle (1L<<7)
#define GCFillStyle (1L<<8)
#define GCFillRule (1L<<9)
#define GCTile (1L<<10)
#define GCStipple (1L<<11)
#define GCTileStipXOrigin (1L<<12)
#define GCTileStipYOrigin (1L<<13)
#define GCFont (1L<<14)
#define GCSubwindowMode (1L<<15)
typedef struct {
    int function;  /* logical operation */
    unsigned long plane_mask; /* plane mask */
    unsigned long foreground; /* foreground pixel */
    unsigned long background; /* background pixel */
    int line_width; /* line width (in pixels) */
    int line_style; /* LineSolid, LineOnOffDash, LineDoubleDash */
    int cap_style; /* CapNotLast, CapButt, CapRound, CapProjecting */
    int join_style; /* JoinMiter, JoinRound, JoinBevel */
    int fill_style; /* FillSolid, FillTiled, FillStippled FillOpaqueStippled */
    int fill_rule; /* EvenOddRule, WindingRule */
    int arc_mode; /* ArcChord, ArcPieSlice */
    Pixmap tile; /* tile pixmap for tiling operations */
    Pixmap stipple; /* stipple 1 plane pixmap for stippling */
    int ts_x_origin; /* offset for tile or stipple operations */
    int ts_y_origin;
    Font font; /* default text font for text operations */
    int subwindow_mode; /* ClipByChildren, IncludeInferiors */
    Bool graphics_exposures; /* boolean, should exposures be generated */
    int clip_x_origin; /* origin for clipping */
    int clip_y_origin;
    Pixmap clip_mask; /* bitmap clipping; other calls for rects */
    int dash_offset; /* pattemed/dashed line information */
    char dashes;
} XGCValues;

The default GC values are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>GXcopy</td>
</tr>
<tr>
<td>plane_mask</td>
<td>All ones</td>
</tr>
<tr>
<td>foreground</td>
<td>0</td>
</tr>
<tr>
<td>background</td>
<td>1</td>
</tr>
<tr>
<td>line_width</td>
<td>0</td>
</tr>
<tr>
<td>line_style</td>
<td>LineSolid</td>
</tr>
<tr>
<td>cap_style</td>
<td>CapButt</td>
</tr>
<tr>
<td>join_style</td>
<td>JoinMiter</td>
</tr>
<tr>
<td>fill_style</td>
<td>FillSolid</td>
</tr>
<tr>
<td>fill_rule</td>
<td>EvenOddRule</td>
</tr>
<tr>
<td>arc_mode</td>
<td>ArcPieSlice</td>
</tr>
<tr>
<td>tile</td>
<td>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)</td>
</tr>
<tr>
<td>stipple</td>
<td>Pixmap of unspecified size filled with ones</td>
</tr>
<tr>
<td>Component</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>ts_x_origin</td>
<td>0</td>
</tr>
<tr>
<td>ts_y_origin</td>
<td>0</td>
</tr>
<tr>
<td>font</td>
<td>&lt;implementation dependent&gt;</td>
</tr>
<tr>
<td>subwindow_mode</td>
<td>ClipByChildren</td>
</tr>
<tr>
<td>graphics_exposures</td>
<td>True</td>
</tr>
<tr>
<td>clip_x_origin</td>
<td>0</td>
</tr>
<tr>
<td>clip_y_origin</td>
<td>0</td>
</tr>
<tr>
<td>clip_mask</td>
<td>None</td>
</tr>
<tr>
<td>dash_offset</td>
<td>0</td>
</tr>
<tr>
<td>dashes</td>
<td>4 (that is, the list [4, 4])</td>
</tr>
</tbody>
</table>

Note that foreground and background are not set to any values likely to be useful in a window.

The function attributes of a GC are used when you update a section of a drawable (the destination) with bits from somewhere else (the source). The function in a GC defines how the new destination bits are to be computed from the source bits and the old destination bits. GXcopy is typically the most useful because it will work on a color display, but special applications may use other functions, particularly in concert with particular planes of a color display. The 16 GC functions, defined in `<X11/X.h>`, are:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Value</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GXclear</td>
<td>0x0</td>
<td>0</td>
</tr>
<tr>
<td>GXand</td>
<td>0x1</td>
<td>src AND dst</td>
</tr>
<tr>
<td>GXandReverse</td>
<td>0x2</td>
<td>src AND NOT dst</td>
</tr>
<tr>
<td>GXcopy</td>
<td>0x3</td>
<td>src</td>
</tr>
<tr>
<td>GXandInverted</td>
<td>0x4</td>
<td>(NOT src) AND dst</td>
</tr>
<tr>
<td>GXnoop</td>
<td>0x5</td>
<td>dst</td>
</tr>
<tr>
<td>GXxor</td>
<td>0x6</td>
<td>src XOR dst</td>
</tr>
<tr>
<td>GXor</td>
<td>0x7</td>
<td>src OR dst</td>
</tr>
<tr>
<td>GXnor</td>
<td>0x8</td>
<td>(NOT src) AND (NOT dst)</td>
</tr>
<tr>
<td>GXequiv</td>
<td>0x9</td>
<td>(NOT src) XOR dst</td>
</tr>
<tr>
<td>GXinvert</td>
<td>0xa</td>
<td>NOT dst</td>
</tr>
<tr>
<td>GXorReverse</td>
<td>0xb</td>
<td>src OR (NOT dst)</td>
</tr>
<tr>
<td>GXcopyInverted</td>
<td>0xc</td>
<td>NOT src</td>
</tr>
<tr>
<td>GXorInverted</td>
<td>0xd</td>
<td>(NOT src) OR dst</td>
</tr>
<tr>
<td>GXnand</td>
<td>0xe</td>
<td>(NOT src) OR (NOT dst)</td>
</tr>
<tr>
<td>GXset</td>
<td>0xf</td>
<td>1</td>
</tr>
</tbody>
</table>

Many graphics operations depend on either pixel values or planes in a GC. The planes attribute is of type long, and it specifies which planes of the destination are to be modified, one bit per plane. A monochrome display has only one plane and will be the least-significant bit of the word. As planes are added to the display hardware, they will occupy more significant bits in the plane mask.

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. A macro constant AllPlanes can be used to refer to all planes of the screen simultaneously. The result is computed by the following:

```
((src FUNC dst) AND plane-mask) OR (dst AND (NOT plane-mask))
```
Range checking is not performed on the values for foreground, background, or plane_mask. They are simply truncated to the appropriate number of bits. The line-width is measured in pixels and either can be greater than or equal to one (wide line) or can be the special value zero (thin line).

Wide lines are drawn centered on the path described by the graphics request. Unless otherwise specified by the join-style or cap-style, the bounding box of a wide line with endpoints \([x_1, y_1], [x_2, y_2]\) and width \(w\) is a rectangle with vertices at the following real coordinates:

\[
[x_1-(w*\sin\theta/2), y_1+(w*\cos\theta/2)], \quad [x_1+(w*\sin\theta/2), y_1-(w*\cos\theta/2)],
\]

\[
[x_2-(w*\sin\theta/2), y_2+(w*\cos\theta/2)], \quad [x_2+(w*\sin\theta/2), y_2-(w*\cos\theta/2)]
\]

Here \(\sin\theta\) is the sine of the angle of the line, and \(\cos\theta\) is the cosine of the angle of the line. A pixel is part of the line and so is drawn if the center of the pixel is fully inside the bounding box (which is viewed as having infinitely thin edges). If the center of the pixel is exactly on the bounding box, it is part of the line if and only if the interior is immediately to its right (\(x\) increasing direction) and the interior or the boundary is immediately to the right (\(x\) increasing direction).

Thin lines (zero line-width) are one-pixel-wide lines drawn using an unspecified, device-dependent algorithm. There are only two constraints on this algorithm.

1. If a line is drawn unclipped from \([x_1,y_1]\) to \([x_2,y_2]\) and if another line is drawn unclipped from \([x_1+dx,y_1+dy]\) to \([x_2+dx,y_2+dy]\), a point \([x,y]\) is touched by drawing the first line if and only if the point \([x+dx,y+dy]\) is touched by drawing the second line.

2. The effective set of points comprising a line cannot be affected by clipping. That is, a point is touched in a clipped line if and only if the point lies inside the clipping region and the point would be touched by the line when drawn unclipped.

A wide line drawn from \([x_1,y_1]\) to \([x_2,y_2]\) always draws the same pixels as a wide line drawn from \([x_2,y_2]\) to \([x_1,y_1]\), not counting cap-style and join-style. It is recommended that this property be true for thin lines, but this is not required. A line-width of zero may differ from a line-width of one in which pixels are drawn. This permits the use of many manufacturers’ line drawing hardware, which may run many times faster than the more precisely specified wide lines.

In general, drawing a thin line will be faster than drawing a wide line of width one. However, because of their different drawing algorithms, thin lines may not mix well aesthetically with wide lines. If it is desirable to obtain precise and uniform results across all displays, a client should always use a line-width of one rather than a line-width of zero.

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

- **LineSolid**: The full path of the line is drawn.
- **LineDoubleDash**: The full path of the line is drawn, but the even dashes are filled differently than the odd dashes (see fill-style) with CapButt style used where even and odd dashes meet.
- **LineOnOffDash**: Only the even dashes are drawn, and cap-style applies to all internal ends of the individual dashes, except CapNotLast is treated as CapButt.

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

- **CapNotLast**: This is equivalent to CapButt except that for a line-width of zero the final endpoint is not drawn.
- **CapButt**: The line is square at the endpoint (perpendicular to the slope of the line) with no projection beyond.
### CapRound
The line has a circular arc with the diameter equal to the line-width, centered on the endpoint. (This is equivalent to CapButt for line-width of zero).

### CapProjecting
The line is square at the end, but the path continues beyond the endpoint for a distance equal to half the line-width. (This is equivalent to CapButt for line-width of zero).

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

<table>
<thead>
<tr>
<th>JoinStyle</th>
<th>CapStyle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JoinMiter</td>
<td>thin</td>
<td>The outer edges of two lines extend to meet at an angle. However, if the angle is less than 11 degrees, then a JoinBevel join-style is used instead.</td>
</tr>
<tr>
<td>JoinRound</td>
<td>thin</td>
<td>The corner is a circular arc with the diameter equal to the line-width, centered on the joinpoint.</td>
</tr>
<tr>
<td>JoinBevel</td>
<td>thin</td>
<td>The corner has CapButt endpoint styles with the triangular notch filled.</td>
</tr>
</tbody>
</table>

For a line with coincident endpoints \((x_1=x_2, y_1=y_2)\), when the cap-style is applied to both endpoints, the semantics depends on the line-width and the cap-style:

<table>
<thead>
<tr>
<th>CapStyle</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapNotLast</td>
<td>thin</td>
<td>The results are device-dependent, but the desired effect is that nothing is drawn.</td>
</tr>
<tr>
<td>CapButt</td>
<td>thin</td>
<td>The results are device-dependent, but the desired effect is that a single pixel is drawn.</td>
</tr>
<tr>
<td>CapRound</td>
<td>thin</td>
<td>The results are the same as for CapButt/thin.</td>
</tr>
<tr>
<td>CapProjecting</td>
<td>thin</td>
<td>The results are the same as for CapButt/thin.</td>
</tr>
<tr>
<td>CapButt</td>
<td>wide</td>
<td>Nothing is drawn.</td>
</tr>
<tr>
<td>CapRound</td>
<td>wide</td>
<td>The closed path is a circle, centered at the endpoint, and with the diameter equal to the line-width.</td>
</tr>
<tr>
<td>CapProjecting</td>
<td>wide</td>
<td>The closed path is a square, aligned with the coordinate axes, centered at the endpoint, and with the sides equal to the line-width.</td>
</tr>
</tbody>
</table>

For a line with coincident endpoints \((x_1=x_2, y_1=y_2)\), 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 represents an infinite 2D plane, with the tile/stipple replicated in all dimensions. When that plane is superimposed on the drawable for use in a graphics operation, the upper left corner of some instance of the tile/stipple is at the coordinates within the drawable specified by the tile/stipple origin. 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 GC, or a BadMatch error results. The stipple pixmap must have depth one and must have the same root as the GC, or a BadMatch error results. For stipple operations where the fill-style is FillStippled but not FillOpaqueStippled, the stipple pattern is tiled in a single plane and acts as an additional clip mask to be ANDed with the clip-mask. Although some sizes may be faster to use than others, any size pixmap can be used for tiling or stippling.

The fill-style defines the contents of the source for line, text, and fill requests. For all text and fill requests (for example, XDrawText, XDrawText16, XFillRectangle, XFillPolygon, and XFillArc); for line requests with line-style LineSolid (for example, XDrawLine).
XDrawSegments, XDrawRectangle, XDrawArc); and for the even dashes for line requests with line-style LineOnOffDash or LineDoubleDash, the following apply:

- **FillSolid**: Foreground
- **FillTiled**: Tile
- **FillOpaqueStippled**: A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one
- **FillStippled**: Foreground masked by stipple

When drawing lines with line-style LineDoubleDash, the odd dashes are controlled by the fill-style in the following manner:

- **FillSolid**: Background
- **FillTiled**: Same as for even dashes
- **FillOpaqueStippled**: Same as for even dashes
- **FillStippled**: Background masked by stipple

Storing a pixmap in a GC 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 GC. If the pixmap is used simultaneously in a graphics request both as a destination and as a tile or stipple, the results are undefined.

For optimum performance, you should draw as much as possible with the same GC (without changing its components). The costs of changing GC components relative to using different GCs depend upon the display hardware and the server implementation. It is quite likely that some amount of GC information will be cached in display hardware and that such hardware can only cache a small number of GCs.

The dashes value is actually a simplified form of the more general patterns that can be set with XSetDashes. Specifying a value of N is equivalent to specifying the two-element list [N, N] in XSetDashes. The value must be nonzero, or a BadValue error results.

The clip-mask restricts writes to the destination drawable. If the clip-mask is set to a pixmap, it must have depth one and have the same root as the GC, or a BadMatch error results. If clip-mask is set to None, the pixels are always drawn regardless of the clip origin. The clip-mask also can be set by calling the XSetClipRectangles or XSetRegion functions. Only pixels where the clip-mask has a bit set to 1 are drawn. Pixels are not drawn outside the area covered by the clip-mask or where the clip-mask has a bit set to 0. The clip-mask affects all graphics requests. The clip-mask does not clip sources. The clip-mask origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

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

The fill-rule defines what pixels are inside (drawn) for paths given in XFillPolygon requests and can be set to EvenOddRule or WindingRule. For EvenOddRule, a point is inside if an infinite ray with the point as origin crosses the path an odd number of times. For WindingRule, a point is inside if an infinite ray with the point as origin crosses an unequal number of clockwise and counterclockwise directed path segments. A clockwise directed path segment is one that crosses the ray from left to right as observed from the point. A
counterclockwise 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 you can simply choose a different ray that is not coincident with a segment.

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

The arc-mode controls filling in the XFillArcs function and can be set to ArcPieSlice or ArcChord. For ArcPieSlice, the arcs are pie-slice filled. For ArcChord, the arcs are chord filled.

The graphics-exposure flag controls GraphicsExpose event generation for XCopyArea and XCopyPlane requests (and any similar requests defined by extensions).

To create a new GC that is usable on a given screen with a depth of drawable, use XCreateGC.

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

- `display`: Specifies the connection to the X server.
- `d`: Specifies the drawable.
- `valuemask`: Specifies which components in the GC are to be set using the information in the specified values structure. This argument is the bitwise inclusive OR of zero or more of the valid GC component mask bits.
- `values`: Specifies any values as specified by the valuemask.

The XCreateGC function creates a graphics context and returns a GC. The GC can be used with any destination drawable having the same root and depth as the specified drawable. Use with other drawables results in a BadMatch error.

XCreateGC can generate BadAlloc, BadDrawable, BadFont, BadMatch, BadPixmap, and BadValue errors.

To copy components from a source GC to a destination GC, use XCopyGC.

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

- `display`: Specifies the connection to the X server.
- `src`: Specifies the components of the source GC.
- `valuemask`: Specifies which components in the GC are to be copied to the destination GC. This argument is the bitwise inclusive OR of zero or more of the valid GC component mask bits.
- `dest`: Specifies the destination GC.

The XCopyGC function copies the specified components from the source GC to the destination GC. The source and destination GCs must have the same root and depth, or a BadMatch error results. The valuemask specifies which component to copy, as for XCreateGC.
XCopyGC can generate BadAlloc, BadGC, and BadMatch errors.

To change the components in a given GC, use XChangeGC.

\[
\text{XChangeGC}(\text{display}, \text{gc}, \text{valuemask}, \text{values})
\]

Display *display;
GC gc;
unsigned long valuemask;
XGCValues *values;

\text{display} \quad \text{Specifies the connection to the X server.}
\text{gc} \quad \text{Specifies the GC.}
\text{valuemask} \quad \text{Specifies which components in the GC are to be changed using information in the specified values structure. This argument is the bitwise inclusive OR of zero or more of the valid GC component mask bits.}
\text{values} \quad \text{Specifies any values as specified by the valuemask.}

The XChangeGC function changes the components specified by valuemask for the specified GC. The values argument contains the values to be set. The values and restrictions are the same as for XCreateGC. Changing the clip-mask overrides any previous XSetClipRectangles request on the context. Changing the dash-offset or dash-list overrides any previous XSetDashes request on the context. The order in which components are verified and altered is server-dependent. If an error is generated, a subset of the components may have been altered.

XChangeGC can generate BadAlloc, BadFont, BadGC, BadMatch, BadPixmap, and BadValue errors.

To obtain components of a given GC, use XGetGCValues.

\[
\text{Status XGetGCValues}(\text{display}, \text{gc}, \text{valuemask}, \text{values_return})
\]

Display *display;
GC gc;
unsigned long valuemask;
XGCValues *values_return;

\text{display} \quad \text{Specifies the connection to the X server.}
\text{gc} \quad \text{Specifies the GC.}
\text{valuemask} \quad \text{Specifies which components in the GC are to be returned in the values_return argument. This argument is the bitwise inclusive OR of zero or more of the valid GC component mask bits.}
\text{values return} \quad \text{Returns the GC values in the specified XGCValues structure.}

The XGetGCValues function returns the components specified by valuemask for the specified GC. If the valuemask contains a valid set of GC mask bits (GCFunction, GCPlaneMask, GCForeground, GCBackground, GCLineWidth, GCLineStyle, GCCapStyle, GCJoinStyle, GCFillStyle, GCFillRule, GCTile, GCStipple, GCTileStipXOrigin, GCTileStipYOrigin, GCFont, GCSubwindowMode, GCGraphicsExposures, GClipXOrigin, GClipYOrigin, GCDashOffset, or GCArcMode) and no error occur, XGetGCValues sets the requested components in values_return and returns a nonzero status. Otherwise, it returns a zero status. Note that the clip-mask and dash-list (represented by the GCClipMask and GCDashList bits, respectively, in the valuemask) cannot be requested. Also note that an invalid resource ID (with one or more of the three most-significant bits set to one) will be returned for GCFont, GCTile, and GCStipple if the component has never been explicitly set by the client.

To free a given GC, use XFreeGC.
XFreeGC(display, gc)
  Display *display;
  GC gc;

display Specifies the connection to the X server.
gc Specifies the GC.
The XFreeGC function destroys the specified GC as well as all the associated storage.
XFreeGC can generate a BadGC error.

To obtain the GContext resource ID for a given GC, use XGContextFromGC.
GContext XGContextFromGC(gc)
  GC gc;
  gc Specifies the GC for which you want the resource ID.

Xlib normally defers sending changes to the components of a GC to the server until a graphics
function is actually called with that GC. This permits batching of component changes into a
single server request. In some circumstances, however, it may be necessary for the client to
explicitly force sending of the changes to the server. An example might be when a protocol
extension uses the GC indirectly, in such a way that the extension interface cannot know what
GC will be used. To force sending of GC component changes, use XFlushGC.

void XFlushGC(display, gc)
  Display *display;
  GC gc;

display Specifies the connection to the X server.
gc Specifies the GC.

7.2. Using GC Convenience Routines
This section discusses how to set the:
• Foreground, background, plane mask, or function components
• Line attributes and dashes components
• Fill style and fill rule components
• Fill tile and stipple components
• Font component
• Clip region component
• Arc mode, subwindow mode, and graphics exposure components

7.2.1. Setting the Foreground, Background, Function, or Plane Mask
To set the foreground, background, plane mask, and function components for a given GC, use
XSetState.

XSetState(display, gc, foreground, background, function, plane_mask)
  Display *display;
  GC gc;
  unsigned long foreground, background;
  int function;
  unsigned long plane_mask;

display Specifies the connection to the X server.
gc Specifies the GC.
foreground  Specifies the foreground you want to set for the specified GC.
background  Specifies the background you want to set for the specified GC.
function    Specifies the function you want to set for the specified GC.
plane_mask  Specifies the plane mask.

XSetState can generate BadAlloc, BadGC, and BadValue errors.

To set the foreground of a given GC, use XSetForeground.

XSetForeground(display, gc, foreground)

Display *display;
GC gc;
unsigned long foreground;

display  Specifies the connection to the X server.
gc       Specifies the GC.
foreground Specifies the foreground you want to set for the specified GC.

XSetForeground can generate BadAlloc and BadGC errors.

To set the background of a given GC, use XSetBackground.

XSetBackground(display, gc, background)

Display *display;
GC gc;
unsigned long background;

display  Specifies the connection to the X server.
gc       Specifies the GC.
background Specifies the background you want to set for the specified GC.

XSetBackground can generate BadAlloc and BadGC errors.

To set the display function in a given GC, use XSetFunction.

XSetFunction(display, gc, function)

Display *display;
GC gc;
int function;

display  Specifies the connection to the X server.
gc       Specifies the GC.
function Specifies the function you want to set for the specified GC.

XSetFunction can generate BadAlloc, BadGC, and BadValue errors.

To set the plane mask of a given GC, use XSetPlaneMask.

XSetPlaneMask(display, gc, plane_mask)

Display *display;
GC gc;
unsigned long plane_mask;

display  Specifies the connection to the X server.
gc       Specifies the GC.
plane_mask Specifies the plane mask.
XSetPlaneMask can generate BadAlloc and BadGC errors.

7.2.2. Setting the Line Attributes and Dashes
To set the line drawing components of a given GC, use XSetLineAttributes.

XSetLineAttributes(display, gc, line_width, line_style, cap_style, join_style)

Display *display;
GC gc;
unsigned int line_width;
int line_style;
int cap_style;
int join_style;

display Specifies the connection to the X server.
gc Specifies the GC.
line_width Specifies the line-width you want to set for the specified GC.
line_style Specifies the line-style you want to set for the specified GC. You can pass LineSolid, LineOnOffDash, or LineDoubleDash.
cap_style Specifies the line-style and cap-style you want to set for the specified GC.
You can pass CapNotLast, CapButt, CapRound, or CapProjecting.
join_style Specifies the line join-style you want to set for the specified GC. You can pass JoinMiter, JoinRound, or JoinBevel.

XSetLineAttributes can generate BadAlloc, BadGC, and BadValue errors.

To set the dash-offset and dash-list for dashed line styles of a given GC, use XSetDashes.

XSetDashes(display, gc, dash_offset, dash_list, n)

Display *display;
GC gc;
int dash_offset;
char dash_list[];
int n;

display Specifies the connection to the X server.
gc Specifies the GC.
dash_offset Specifies the phase of the pattern for the dashed line-style you want to set for the specified GC.
dash_list Specifies the dash-list for the dashed line-style you want to set for the specified GC.
n Specifies the number of elements in dash_list.

The XSetDashes function sets the dash-offset and dash-list attributes for dashed line styles in the specified GC. There must be at least one element in the specified dash_list, or a BadValue error results. The initial and alternating elements (second, fourth, and so on) of the dash_list are the even dashes, and the others are the odd dashes. Each element specifies a dash length in pixels. All of the elements must be nonzero, or a BadValue error results. Specifying an odd-length list is equivalent to specifying the same list concatenated with itself to produce an even-length list.

The dash-offset defines the phase of the pattern, specifying how many pixels into the dash-list the pattern should actually begin in any single graphics request. Dashing is continuous through path 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 for 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 135 and 225 degrees from the x axis. For all other lines, the major axis is the y axis.

XSetDashes can generate BadAlloc, BadGC, and BadValue errors.

7.2.3. Setting the Fill Style and Fill Rule
To set the fill-style of a given GC, use XSetFillStyle.

XSetFillStyle(display, gc, fill_style)

Display *display;
GC gc;
int fill_style;

display Specifies the connection to the X server.
gc Specifies the GC.
fill_style Specifies the fill-style you want to set for the specified GC. You can pass FillSolid, FillTiled, FillStippled, or FillOpaqueStippled.

XSetFillStyle can generate BadAlloc, BadGC, and BadValue errors.

To set the fill-rule of a given GC, use XSetFillRule.

XSetFillRule(display, gc, fill_rule)

Display *display;
GC gc;
int fill_rule;

display Specifies the connection to the X server.
gc Specifies the GC.
fill_rule Specifies the fill-rule you want to set for the specified GC. You can pass EvenOddRule or WindingRule.

XSetFillRule can generate BadAlloc, BadGC, and BadValue errors.

7.2.4. Setting the Fill Tile and Stipple
Some displays have hardware support for tiling or stippling with patterns of specific sizes. Tiling and stippling operations that restrict themselves to those specific sizes run much faster than such operations with arbitrary size patterns. Xlib provides functions that you can use to determine the best size, tile, or stipple for the display as well as to set the tile or stipple shape and the tile or stipple origin.

To obtain the best size of a tile, stipple, or cursor, use XQueryBestSize.

Status XQueryBestSize(display, class, which_screen, width, height, width_return, height_return)

Display *display;
int class;
Drawable which_screen;
unsigned int width, height;
unsigned int *width_return, *height_return;

display Specifies the connection to the X server.
class Specifies the class that you are interested in. You can pass TileShape, CursorShape, or StippleShape.
which_screen  Specifies any drawable on the screen.
width
height  Specify the width and height.
width_return
height_return  Return the width and height of the object best supported by the display hardware.

The XQueryBestSize function returns the best or closest size to the specified size. For CursorShape, this is the largest size that can be fully displayed on the screen specified by which_screen. For TileShape, this is the size that can be tiled fastest. For StippleShape, this is the size that can be stippled fastest. For CursorShape, the drawable indicates the desired screen. For TileShape and StippleShape, the drawable indicates the screen and possibly the window class and depth. An InputOnly window cannot be used as the drawable for TileShape or StippleShape, or a BadMatch error results.

XQueryBestSize can generate BadDrawable, BadMatch, and BadValue errors.

To obtain the best fill tile shape, use XQueryBestTile.

Status XQueryBestTile(display, which_screen, width, height, width_return, height_return)

Display *display;
Drawable which_screen;
unsigned int width, height;
unsigned int *width_return, *height_return;

display  Specifies the connection to the X server.
which_screen  Specifies any drawable on the screen.
width
height  Specify the width and height.
width_return
height_return  Return the width and height of the object best supported by the display hardware.

The XQueryBestTile function returns the best or closest size, that is, the size that can be tiled fastest on the screen specified by which_screen. The drawable indicates the screen and possibly the window class and depth. If an InputOnly window is used as the drawable, a BadMatch error results.

XQueryBestTile can generate BadDrawable and BadMatch errors.

To obtain the best stipple shape, use XQueryBestStipple.

Status XQueryBestStipple(display, which_screen, width, height, width_return, height_return)

Display *display;
Drawable which_screen;
unsigned int width, height;
unsigned int *width_return, *height_return;

display  Specifies the connection to the X server.
which_screen  Specifies any drawable on the screen.
width
height  Specify the width and height.
width_return
height_return  Return the width and height of the object best supported by the display hardware.
The `XQueryBestStipple` function returns the best or closest size, that is, the size that can be stippled fastest on the screen specified by `which_screen`. The drawable indicates the screen and possibly the window class and depth. If an `InputOnly` window is used as the drawable, a `BadMatch` error results.

`XQueryBestStipple` can generate `BadDrawable` and `BadMatch` errors.

To set the fill tile of a given GC, use `XSetTile`.

```c
XSetTile(display, gc, tile)
    Display *display;
    GC gc;
    Pixmap tile;
```

- `display` Specifies the connection to the X server.
- `gc` Specifies the GC.
- `tile` Specifies the fill tile you want to set for the specified GC.

The tile and GC must have the same depth, or a `BadMatch` error results.

`XSetTile` can generate `BadAlloc`, `BadGC`, `BadMatch`, and `BadPixmap` errors.

To set the stipple of a given GC, use `XSetStipple`.

```c
XSetStipple(display, gc, stipple)
    Display *display;
    GC gc;
    Pixmap stipple;
```

- `display` Specifies the connection to the X server.
- `gc` Specifies the GC.
- `stipple` Specifies the stipple you want to set for the specified GC.

The stipple must have a depth of one, or a `BadMatch` error results.

`XSetStipple` can generate `BadAlloc`, `BadGC`, `BadMatch`, and `BadPixmap` errors.

To set the tile or stipple origin of a given GC, use `XSetTSOrigin`.

```c
XSetTSOrigin(display, gc, ts_x_origin, ts_y_origin)
    Display *display;
    GC gc;
    int ts_x_origin, ts_y_origin;
```

- `display` Specifies the connection to the X server.
- `gc` Specifies the GC.
- `ts_x_origin` Specifying the x coordinate of the tile and stipple origin.
- `ts_y_origin` Specify the y coordinate of the tile and stipple origin.

When graphics requests call for tiling or stippling, the parent’s origin will be interpreted relative to whatever destination drawable is specified in the graphics request.

`XSetTSOrigin` can generate `BadAlloc` and `BadGC` error.

### 7.2.5. Setting the Current Font

To set the current font of a given GC, use `XSetFont`.

```c
XSetFont(display, gc, fonts)
    Display *display;
    GC gc;
    Pixmap fonts;
```

- `display` Specifies the connection to the X server.
- `gc` Specifies the GC.
- `fonts` Specifies the font you want to set for the specified GC.

The font must have a depth of one, or a `BadMatch` error results.

`XSetFont` can generate `BadAlloc`, `BadGC`, `BadMatch`, and `BadPixmap` errors.
XSetFont(display, gc, font)
    Display *display;
    GC gc;
    Font font;

display        Specifies the connection to the X server.
gc             Specifies the GC.
font           Specifies the font.

XSetFont can generate BadAlloc, BadFont, and BadGC errors.

7.2.6. Setting the Clip Region

Xlib provides functions that you can use to set the clip-origin and the clip-mask or set the
clip-mask to a list of rectangles.

To set the clip-origin of a given GC, use XSetClipOrigin.

XSetClipOrigin(display, gc, clip_x_origin, clip_y_origin)
    Display *display;
    GC gc;
    int clip_x_origin, clip_y_origin;

display      Specifies the connection to the X server.
gc           Specifies the GC.
clip_x_origin Specify the x and y coordinates of the clip-mask origin.
clip_y_origin

The clip-mask origin is interpreted relative to the origin of whatever destination drawable is
specified in the graphics request.

XSetClipOrigin can generate BadAlloc and BadGC errors.

To set the clip-mask of a given GC to the specified pixmap, use XSetClipMask.

XSetClipMask(display, gc, pixmap)
    Display *display;
    GC gc;
   Pixmap pixmap;

display      Specifies the connection to the X server.
gc           Specifies the GC.
pixmap      Specifies the pixmap or None.

If the clip-mask is set to None, the pixels are are always drawn (regardless of the clip-origin).

XSetClipMask can generate BadAlloc, BadGC, BadMatch, and BadPixmap errors.

To set the clip-mask of a given GC to the specified list of rectangles, use XSetClipRectan-
gles.

XSetClipRectangles(display, gc, clip_x_origin, clip_y_origin, rectangles, n, ordering)
    Display *display;
    GC gc;
    int clip_x_origin, clip_y_origin;
    XRectangle rectangles[];
    int n;
    int ordering;
Xlib - C Library

XI1, Release 5

display  Specifies the connection to the X server.
gc  Specifies the GC.
clip_x_origin  Specifies the x and y coordinates of the clip-mask origin.
clip_y_origin  Specifies an array of rectangles that define the clip-mask.
rectangles  Specifies the number of rectangles.
n  Specifies the ordering relations on the rectangles. You can pass Unsorted, YSorted, YXSorted, or YXBanded.

The XSetClipRectangles function changes the clip-mask in the specified GC to the specified list of rectangles and sets the clip origin. The output is 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 the graphics results will be undefined. Note that the list of rectangles can be empty, which effectively disables output. This is the opposite of passing None as the clip-mask in XCreateGC, XChangeGC, and XSetClipMask.

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

XSetClipRectangles can generate BadAlloc, BadGC, BadMatch, and BadValue errors.

Xlib provides a set of basic functions for performing region arithmetic. For information about these functions, see section 16.5.

7.2.7. Setting the Arc Mode, Subwindow Mode, and Graphics Exposure

To set the arc mode of a given GC, use XSetArcMode.

XSetArcMode(display, gc, arc_mode)
    Display *display;
    GC gc;
    int arc_mode;

display  Specifies the connection to the X server.
gc  Specifies the GC.
arce_mode  Specifies the arc mode. You can pass ArcChord or ArcPieSlice.

XSetArcMode can generate BadAlloc, BadGC, and BadValue errors.

To set the subwindow mode of a given GC, use XSetSubwindowMode.

XSetSubwindowMode(display, gc, subwindow_mode)
    Display *display;
    GC gc;
    int subwindow_mode;

display  Specifies the connection to the X server.
gc  Specifies the GC.
subwindow_mode

Specifies the subwindow mode. You can pass ClipByChildren or IncludeIn-
fieriors.

XSetSubwindowMode can generate BadAlloc, BadGC, and BadValue errors.

To set the graphics-exposures flag of a given GC, use XSetGraphicsExposures.

XSetGraphicsExposures(display, gc, graphics_exposures)

Display *display;
GC gc;
Bool graphics_exposures;

display Specifies the connection to the X server.
 gc Specifies the GC.
 graphics_exposures Specifies a Boolean value that indicates whether you want GraphicsExpose and NoExpose events to be reported when calling XCopyArea and XCopy-
 Plane with this GC.

XSetGraphicsExposures can generate BadAlloc, BadGC, and BadValue errors.
Chapter 8
Graphics Functions

Once you have established a connection to a display, you can use the Xlib graphics functions to:
- Clear and copy areas
- Draw points, lines, rectangles, and arcs
- Fill areas
- Manipulate fonts
- Draw text
- Transfer images between clients and the server

If the same drawable and GC is used for each call, Xlib batches back-to-back calls to XDrawPoint, XDrawLine, XDrawRectangle, XFillArc, and XFillRectangle. Note that this reduces the total number of requests sent to the server.

8.1. Clearing Areas

Xlib provides functions that you can use to clear an area or the entire window. Because pixmaps do not have defined backgrounds, they cannot be filled by using the functions described in this section. Instead, to accomplish an analogous operation on a pixmap, you should use XFillRectangle, which sets the pixmap to a known value.

To clear a rectangular area of a given window, use XClearArea.

XClearArea(display, w, x, y, width, height, exposures)

Display *display;
Window w;
int x, y;
unsigned int width, height;
Bool exposures;

display Specifies the connection to the X server.
w Specifies the window.
x y Specify the x and y coordinates, which are relative to the origin of the window and specify the upper-left corner of the rectangle.
width height Specify the width and height, which are the dimensions of the rectangle.
exposures Specifies a Boolean value that indicates if Expose events are to be generated.

The XClearArea function paints a rectangular area in the specified window according to the specified dimensions with the window’s background pixel or pixmap. The subwindow-mode effectively is ClipByChildren. 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 clipped by any children is filled with this tile. If the window has background None, the contents of the window are not changed. In either case, if exposures is True, one or more Expose events are generated for regions of the rectangle that are either visible or are being retained in a backing store. If you specify a window whose class is InputOnly, a BadMatch error results.

118
XClearArea can generate BadMatch, BadValue, and BadWindow errors.

To clear the entire area in a given window, use XClearWindow.
\[
\text{XClearWindow}(\text{display, } w) \\
\quad \text{Display } *\text{display}; \\
\quad \text{Window } w;
\]
- \textit{display} Specifies the connection to the X server.
- \textit{w} Specifies the window.

The XClearWindow function clears the entire area in the specified window and is equivalent to XClearArea (display, w, 0, 0, 0, False). If the window has a defined background tile, the rectangle is tiled with a plane-mask of all ones and GXcopy function. If the window has background None, the contents of the window are not changed. If you specify a window whose class is InputOnly, a BadMatch error results.

XClearWindow can generate BadMatch and BadWindow errors.

8.2. Copying Areas

Xlib provides functions that you can use to copy an area or a bit plane.

To copy an area between drawables of the same root and depth, use XCopyArea.
\[
\text{XCopyArea}(\text{display, } src, \text{ dest, } gc, \text{ src}_x, \text{ src}_y, \text{ width, } \text{ height, } \text{ dest}_x, \text{ dest}_y) \\
\quad \text{Display } *\text{display}; \\
\quad \text{Drawable } src, \text{ dest}; \\
\quad \text{GC } gc; \\
\quad \text{int } src_x, src_y; \\
\quad \text{unsigned int } width, height; \\
\quad \text{int } dest_x, dest_y;
\]
- \textit{display} Specifies the connection to the X server.
- \textit{src} \textit{dest} Specify the source and destination rectangles to be combined.
- \textit{gc} Specifies the GC.
- \textit{src}_x \textit{src}_y Specify the x and y coordinates, which are relative to the origin of the source rectangle and specify its upper-left corner.
- \textit{width} \textit{height} Specify the width and height, which are the dimensions of both the source and destination rectangles.
- \textit{dest}_x \textit{dest}_y Specify the x and y coordinates, which are relative to the origin of the destination rectangle and specify its upper-left corner.

The XCopyArea function combines the specified rectangle of src with the specified rectangle of dest. The drawables must have the same root and depth, or a BadMatch 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, those regions are not copied. Instead, the following occurs on all corresponding destination regions that are either visible or are retained in backing store. If the destination is a window with a background other than None, corresponding regions of the destination are tiled with that background (with plane-mask of all ones and GXcopy function). Regardless of tiling or whether the destination is a window or a pixmap, if graphics-exposures is True, then GraphicsExpose events for all corresponding destination regions are generated. If graphics-exposures is True but no
GraphicsExpose events are generated, a NoExpose event is generated. Note that by default graphics-exposures is True in new GCs.

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

XCopyArea can generate BadDrawable, BadGC, and BadMatch errors.

To copy a single bit plane of a given drawable, use XCopyPlane.

\[
\text{XCopyPlane}(\text{display}, \text{src}, \text{dest}, \text{gc}, \text{src}_x, \text{src}_y, \text{width}, \text{height}, \text{dest}_x, \text{dest}_y, \text{plane})
\]

- \text{display} Specifies the connection to the X server.
- \text{src} Specify the source and destination rectangles to be combined.
- \text{dest} Specifies the GC.
- \text{src}_x Specify the x and y coordinates, which are relative to the origin of the source rectangle and specify its upper-left corner.
- \text{src}_y Specify the width and height, which are the dimensions of both the source and destination rectangles.
- \text{dest}_x Specify the x and y coordinates, which are relative to the origin of the destination rectangle and specify its upper-left corner.
- \text{dest}_y Specify the bit plane. You must set exactly one bit to 1.

The XCopyPlane function uses a single bit plane of the specified source rectangle combined with the specified GC to modify the specified rectangle of dest. The drawables must have the same root but need not have the same depth. If the drawables do not have the same root, a BadMatch error results. If plane does not have exactly one bit set to 1 and the values of planes must be less than \(2^n\), where \(n\) is the depth of src, a BadValue error results.

Effectively, XCopyPlane forms a pixmap of the same depth as the rectangle of dest and with a size specified by the source region. It uses the foreground/background pixels in the GC (foreground everywhere the bit plane in src contains a bit set to 1, background everywhere the bit plane in src contains a bit set to 0) and the equivalent of a CopyArea protocol request is performed with all the same exposure semantics. This can also be thought of as using the specified region of the source bit plane as a stipple with a fill-style of FillOpaqueStippled for filling a rectangular area of the destination.

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

XCopyPlane can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

8.3. Drawing Points, Lines, Rectangles, and Arcs

Xlib provides functions that you can use to draw:
- A single point or multiple points
• A single line or multiple lines
• A single rectangle or multiple rectangles
• A single arc or multiple arcs

Some of the functions described in the following sections use these structures:

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

typedef struct {
  short x, y;
} XPoint;

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

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

All x and y members are signed integers. The width and height members are 16-bit unsigned integers. You should be careful not to generate coordinates and sizes out of the 16-bit ranges, because the protocol only has 16-bit fields for these values.

### 8.3.1. Drawing Single and Multiple Points

To draw a single point in a given drawable, use **XDrawPoint**.

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

display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
x y Specify the x and y coordinates where you want the point drawn.

To draw multiple points in a given drawable, use **XDrawPoints**.

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

display Specifies the connection to the X server.
d Specifies the drawable.

gc Specifies the GC.

points Specifies an array of points.

npoints Specifies the number of points in the array.

mode Specifies the coordinate mode. You can pass CoordModeOrigin or CoordModePrevious.

The XDrawPoint function uses the foreground pixel and function components of the GC to draw a single point into the specified drawable; XDrawPoints draws multiple points this way. CoordModeOrigin treats all coordinates as relative to the origin, and CoordModePrevious treats all coordinates after the first as relative to the previous point. XDrawPoints draws the points in the order listed in the array.

Both functions use these GC components: function, plane-mask, foreground, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask.

XDrawPoint can generate BadDrawable, BadGC, and BadMatch errors. XDrawPoints can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

8.3.2. Drawing Single and Multiple Lines

To draw a single line between two points in a given drawable, use XDrawLine.

XDrawLine(display, d, gc, x1, y1, x2, y2)

Display *display;
Drawable d;
GC gc;
int x1, y1, x2, y2;

display Specifies the connection to the X server.

d Specifies the drawable.

gc Specifies the GC.

x1 y1 x2 y2 Specify the points. (x1, y1) and (x2, y2) to be connected.

To draw multiple lines in a given drawable, use XDrawLines.

XDrawLines(display, d, gc, points, npoints, mode)

Display *display;
Drawable d;
GC gc;
XPoint *points;
int npoints;
int mode;

display Specifies the connection to the X server.

d Specifies the drawable.

gc Specifies the GC.

points Specifies an array of points.

npoints Specifies the number of points in the array.

mode Specifies the coordinate mode. You can pass CoordModeOrigin or CoordModePrevious.
To draw multiple, unconnected lines in a given drawable, use XDrawSegments.

\[
\text{XDrawSegments}(display, d, gc, segments, nsegments)
\]

\[
\begin{align*}
\text{Display} & \quad \text{display} ; \\
\text{Drawable} & \quad d ; \\
\text{GC} & \quad gc ; \\
\text{XSegment} & \quad \ast \text{segments} ; \\
\text{int} & \quad nsegments ;
\end{align*}
\]

\[
\begin{align*}
display & \quad \text{Specifies the connection to the X server.} \\
d & \quad \text{Specifies the drawable.} \\
gc & \quad \text{Specifies the GC.} \\
segments & \quad \text{Specifies an array of segments.} \\
nsegments & \quad \text{Specifies the number of segments in the array.}
\end{align*}
\]

The XDrawLine function uses the components of the specified GC to draw a line between the specified set of points \((x_1, y_1)\) and \((x_2, y_2)\). It does not perform joining at coincident endpoints. For any given line, XDrawLine does not draw a pixel more than once. If lines intersect, the intersecting pixels are drawn multiple times.

The XDrawLines function uses the components of the specified GC to draw \(n-1\) lines between each pair of points \((\text{point}[i], \text{point}[i+1])\) in the array of XPoint structures. It draws the lines in the order listed in the array. The lines join correctly at all intermediate points, and if the first and last points coincide, the first and last lines also join correctly. For any given line, XDrawLines does not draw a pixel 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 protocol request were a single, filled shape. CoordModeOrigin treats all coordinates as relative to the origin, and CoordModePrevious treats all coordinates after the first as relative to the previous point.

The XDrawSegments function draws multiple, unconnected lines. For each segment, XDrawSegments draws a line between \((x_1, y_1)\) and \((x_2, y_2)\). It draws the lines in the order listed in the array of XSegment structures and does not perform joining at coincident endpoints. For any given line, XDrawSegments does not draw a pixel more than once. If lines intersect, the intersecting pixels are drawn multiple times.

All three functions use these GC components: function, plane-mask, line-width, line-style, cap-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. The XDrawLines function also uses the join-style GC component. All three functions also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

XDrawLine, XDrawLines, and XDrawSegments can generate BadDrawable, BadGC, and BadMatch errors. XDrawLines also can generate BadValue errors.

### 8.3.3. Drawing Single and Multiple Rectangles

To draw the outline of a single rectangle in a given drawable, use XDrawRectangle.

\[
\text{XDrawRectangle}(display, d, gc, x, y, width, height)
\]

\[
\begin{align*}
\text{Display} & \quad \ast \text{display} ; \\
\text{Drawable} & \quad d ; \\
\text{GC} & \quad gc ; \\
\text{int} & \quad x, y ; \\
\text{unsigned int} & \quad \text{width, height} ;
\end{align*}
\]

\[
\begin{align*}
display & \quad \text{Specifies the connection to the X server.} \\
d & \quad \text{Specifies the drawable.}
\end{align*}
\]
gc Specifies the GC.

x
y Specify the x and y coordinates, which specify the upper-left corner of the rectangle.

width
height Specify the width and height, which specify the dimensions of the rectangle.

To draw the outline of multiple rectangles in a given drawable, use XDrawRectangles.

XDrawRectangles(display, d, gc, rectangles, nrectangles)

Display *display;
Drawable d;
GC gc;
XRectangle rectangles[];
int nrectangles;

display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
rectangles Specifies an array of rectangles.
nrectangles Specifies the number of rectangles in the array.

The XDrawRectangle and XDrawRectangles functions draw the outlines of the specified rectangle or rectangles as if a five-point PolyLine protocol request were specified for each rectangle:

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

For the specified rectangle or rectangles, these functions do not draw a pixel more than once. XDrawRectangles draws the rectangles in the order listed in the array. If rectangles intersect, the intersecting pixels are drawn multiple times.

Both functions use these GC components: function, plane-mask, line-width, line-style, cap-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

XDrawRectangle and XDrawRectangles can generate BadDrawable, BadGC, and BadMatch errors.

8.3.4. Drawing Single and Multiple Arcs

To draw a single arc in a given drawable, use XDrawArc.

XDrawArc(display, d, gc, x, y, width, height, angle1, angle2)

Display *display;
Drawable d;
GC gc;
int x, y;
unsigned int width, height;
int angle1, angle2;

display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
x
Specify the x and y coordinates, which are relative to the origin of the drawable and specify the upper-left corner of the bounding rectangle.

**width**

**height**

Specify the width and height, which are the major and minor axes of the arc.

**angle1**

Specifies the start of the arc relative to the three-o’clock position from the center, in units of degrees * 64.

**angle2**

Specifies the path and extent of the arc relative to the start of the arc, in units of degrees * 64.

To draw multiple arcs in a given drawable, use **XDrawArcs**.

```
XDrawArcs(display, d, gc, arcs, narcs)
```

- **Display** `display`
- **Drawable** `d`
- **GC** `gc`
- **XArc** `arcs`
- **int** `narcs`

**XDrawArc** draws a single circular or elliptical arc, and **XDrawArcs** draws multiple circular or elliptical arcs. Each arc is specified by a rectangle and two angles. The center of the circle or ellipse is the center of the rectangle, and the major and minor axes are specified by the width and height. Positive angles indicate counterclockwise motion, and negative angles indicate clockwise motion. If the magnitude of angle2 is greater than 360 degrees, **XDrawArc** or **XDrawArcs** truncates it to 360 degrees.

For an arc specified as `[x, y, width, height, angle1, angle2]`, the origin of the major and minor axes is at `[x + width/2, y + height/2]`, and the infinitely thin path describing the entire circle or ellipse intersects the horizontal axis at `[x, y + height/2]` and `[x + width, y + height/2]` and intersects the vertical axis at `[x + width/2, y]` and `[x + width/2, y + height]`. These coordinates can be fractional and so 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, width, height, angle1, angle2]`, 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\left(\tan(\text{normal-angle}) \times \frac{\text{width}}{\text{height}}\right) + \text{adjust}
```

The skewed-angle and normal-angle are expressed in radians (rather than in degrees scaled by 64) in the range `[0, 2\pi)` and where atan returns a value in the range `[-\pi/2, \pi/2]` and adjust is:
For any given arc, XDrawArc and XDrawArcs do not draw a pixel more than once. If two arcs join correctly and if the line-width is greater than zero and the arcs intersect, XDrawArc and XDrawArcs do not draw a pixel more than once. Otherwise, the intersecting pixels of intersecting arcs are drawn multiple times. Specifying an arc with one endpoint and a clockwise extent draws the same pixels as specifying the other endpoint and an equivalent counterclockwise extent, except as it affects joins.

If the last point in one arc coincides with the first point in the following arc, the two arcs will join correctly. If the first point in the first arc coincides with the last point in the last arc, the two arcs will join correctly. By specifying one axis to be zero, a horizontal or vertical line can be drawn. Angles are computed based solely on the coordinate system and ignore the aspect ratio.

Both functions use these GC components: function, plane-mask, line-width, line-style, cap-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

XDrawArc and XDrawArcs can generate BadDrawable, BadGC, and BadMatch errors.

8.4. Filling Areas

Xlib provides functions that you can use to fill:

- A single rectangle or multiple rectangles
- A single polygon
- A single arc or multiple arcs

8.4.1. Filling Single and Multiple Rectangles

To fill a single rectangular area in a given drawable, use XFillRectangle.

XFillRectangle(display, d, gc, x, y, width, height)

Display *display;
Drawable d;
GC gc;
int x, y;
unsigned int width, height;

display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
x y Specify the x and y coordinates, which are relative to the origin of the drawable and specify the upper-left corner of the rectangle.
width height Specify the width and height, which are the dimensions of the rectangle to be filled.

To fill multiple rectangular areas in a given drawable, use XFillRectangles.
XFillRectangles(display, d, gc, rectangles, nrectangles)
   Display *display;
   Drawable d;
   GC gc;
   XRectangle *rectangles;
   int nrectangles;
   
   display  Specifies the connection to the X server.
   d        Specifies the drawable.
   gc       Specifies the GC.
   rectangles Specifies an array of rectangles.
   nrectangles Specifies the number of rectangles in the array.

The XFillRectangle and XFillRectangles functions fill the specified rectangle or rectangles as if a four-point FillPolygon protocol request were specified for each rectangle:

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

Each function uses the x and y coordinates, width and height dimensions, and GC you specify.

XFillRectangles fills the rectangles in the order listed in the array. For any given rectangle, XFillRectangle and XFillRectangles do not draw a pixel more than once. If rectangles intersect, the intersecting pixels are drawn multiple times.

Both functions use these GC components: function, plane-mask, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XFillRectangle and XFillRectangles can generate BadDrawable, BadGC, and BadMatch errors.

8.4.2. Filling a Single Polygon

To fill a polygon area in a given drawable, use XFillPolygon.

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

   display  Specifies the connection to the X server.
   d        Specifies the drawable.
   gc       Specifies the GC.
   points   Specifies an array of points.
   npoints  Specifies the number of points in the array.
   shape    Specifies a shape that helps the server to improve performance. You can pass Complex, Convex, or Nonconvex.
   mode     Specifies the coordinate mode. You can pass CoordModeOrigin or CoordModePrevious.

XFillPolygon fills the region closed by the specified path. The path is closed automatically if the last point in the list does not coincide with the first point. XFillPolygon does not draw a pixel of the region more than once. CoordModeOrigin treats all coordinates as relative to the
origin, and CoordModePrevious treats all coordinates after the first as relative to the previous point.

Depending on the specified shape, the following occurs:

- If shape is Complex, the path may self-intersect. Note that contiguous coincident points in the path are not treated as self-intersection.
- If shape is Convex, 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 you specify Convex for a path that is not convex, the graphics results are undefined.
- If shape is Nonconvex, the path does not self-intersect, but the shape is not wholly convex. If known by the client, specifying Nonconvex instead of Complex may improve performance. If you specify Nonconvex for a self-intersecting path, the graphics results are undefined.

The fill-rule of the GC controls the filling behavior of self-intersecting polygons.

This function uses these GC components: function, plane-mask, fill-style, fill-rule, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. It also uses these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XFillPolygon can generate BadDrawable, BadGC, BadMatch, and BadValue errors.

8.4.3. Filling Single and Multiple Arcs

To fill a single arc in a given drawable, use XFillArc.

XFillArc(display, d, gc, x, y, width, height, angle1, angle2)

Display *display;
Drawable d;
GC gc;
int x, y;
unsigned int width, height;
int angle1, angle2;

display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
x y Specify the x and y coordinates, which are relative to the origin of the drawable and specify the upper-left corner of the bounding rectangle.
width height Specify the width and height, which are the major and minor axes of the arc.
angle1 Specifies the start of the arc relative to the three-o’clock position from the center, in units of degrees * 64.
angle2 Specifies the path and extent of the arc relative to the start of the arc, in units of degrees * 64.

To fill multiple arcs in a given drawable, use XFillArcs.

XFillArcs(display, d, gc, arcs, narcs)

Display *display;
Drawable d;
GC gc;
XArc *arcs;
int narcs;
display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
archs Specifies an array of arcs.
narchs Specifies the number of arcs in the array.

For each arc, XFillArc or XFillArcs fills the region closed by the infinitely thin path described by the specified arc and, depending on the arc-mode specified in the GC, one or two line segments. For ArcChord, the single line segment joining the endpoints of the arc is used. For ArcPieSlice, the two line segments joining the endpoints of the arc with the center point are used. XFillArcs fills the arcs in the order listed in the array. For any given arc, XFillArc and XFillArcs do not draw a pixel more than once. If regions intersect, the intersecting pixels are drawn multiple times.

Both functions use these GC components: function, plane-mask, fill-style, arc-mode, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XFillArc and XFillArcs can generate BadDrawable, BadGC, and BadMatch errors.

8.5. Font Metrics

A font is a graphical description of a set of characters that are used to increase efficiency whenever a set of small, similar sized patterns are repeatedly used.

This section discusses how to:

- Load and free fonts
- Obtain and free font names
- Compute character string sizes
- Return logical extents
- Query character string sizes

The X server loads fonts whenever a program requests a new font. The server can cache fonts for quick lookup. Fonts are global across all screens in a server. Several levels are possible when dealing with fonts. Most applications simply use XLoadQueryFont to load a font and query the font metrics.

Characters in fonts are regarded as masks. Except for image text requests, the only pixels modified are those in which bits are set to 1 in the character. This means that it makes sense to draw text using stipples or tiles (for example, many menus gray-out unusable entries).

The XFontStruct structure contains all of the information for the font and consists of the font-specific information as well as a pointer to an array of XCharStruct structures for the characters contained in the font. The XFontStruct, XFontProp, and XCharStruct structures contain:

typedef struct {
  short lbearing; /* origin to left edge of raster */
  short rbearing; /* origin to right edge of raster */
  short width; /* advance to next char’s origin */
  short ascent; /* baseline to top edge of raster */
  short descent; /* baseline to bottom edge of raster */
  unsigned short attributes; /* per char flags (not predefined) */
} XCharStruct;

typedef struct {
  Atom name;
}
unsigned long card32;
} XFontProp;

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

typedef struct {
    XExtData *ext_data;
    Font fid;
    unsigned direction;
    unsigned min_char_or_byte2;
    unsigned max_char_or_byte2;
    unsigned min_byte1;
    unsigned max_byte1;
    Bool all_chars_exist;
    unsigned default_char;
    int n_properties;
    XFontProp *properties;
    XCharStruct min_bounds;
    XCharStruct max_bounds;
    XCharStruct *per_char;
    int ascent;
    int descent;
} XFontStruct;

/* normal 16 bit characters are two bytes */

X supports single byte/character, two bytes/character matrix, and 16-bit character text operations. Note that any of these forms can be used with a font, but a single byte/character text request can only specify a single byte (that is, the first row of a 2-byte font). You should view 2-byte fonts as a two-dimensional matrix of defined characters: byte1 specifies the range of defined rows and byte2 defines the range of defined columns of the font. Single byte/character fonts have one row defined, and the byte2 range specified in the structure defines a range of characters.

The bounding box of a character is defined by the XCharStruct of that character. When characters are absent from a font, the default_char is used. When fonts have all characters of the same size, only the information in the XFontStruct min and max bounds are used.

The members of the XFontStruct have the following semantics:

- The direction member can be either FontLeftToRight or FontRightToLeft. It is just a hint as to whether most XCharStruct elements have a positive (FontLeftToRight) or a negative (FontRightToLeft) character width metric. The core protocol defines no support for vertical text.

- If the min_byte1 and max_byte1 members are both zero, min_char_or_byte2 specifies the linear character index corresponding to the first element of the per_char array, and max_char_or_byte2 specifies the linear character index of the last element.

If either min_byte1 or max_byte1 are nonzero, both min_char_or_byte2 and max_char_or_byte2 are less than 256, and the 2-byte character index values corresponding to the per_char array element N (counting from 0) are:

\[ \text{byte1} = \frac{N}{D} + \text{min_byte1} \]
\[ \text{byte2} = \frac{N}{D} + \text{min_char_or_byte2} \]

where:

\[ D = \text{max_char_or_byte2} - \text{min_char_or_byte2} + 1 \]
\[ / = \text{integer division} \]
• If the per_char pointer is NULL, all glyphs between the first and last character indexes inclusive have the same information, as given by both min_bounds and max_bounds.
• If all_chars_exist is True, all characters in the per_char array have nonzero bounding boxes.
• The default_char member specifies the character that will be used when an undefined or nonexistent character is printed. The default_char is a 16-bit character (not a 2-byte character). 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, no printing is performed for an undefined or nonexistent character.
• The min_bounds and max_bounds members contain the most extreme values of each individual XCharStruct component over all elements of this array (and ignore nonexistent characters). The bounding box of the font (the smallest rectangle enclosing the shape obtained by superimposing all of the characters at the same origin \(x, y\)) has its upper-left coordinate at:

\[
[x + \text{min}\_\text{bounds}.l\text{bearing}, y - \text{max}\_\text{bounds}.\text{ascent}]
\]

Its width is:

\[
\text{max}\_\text{bounds}.r\text{bearing} - \text{min}\_\text{bounds}.l\text{bearing}
\]

Its height is:

\[
\text{max}\_\text{bounds}.\text{ascent} + \text{max}\_\text{bounds}.\text{descent}
\]

• The ascent member is the logical extent of the font above the baseline that is used for determining line spacing. Specific characters may extend beyond this.
• The descent member is the logical extent of the font at or below the baseline that is used for determining line spacing. Specific characters may extend beyond this.
• If the baseline is at Y-coordinate \(y\), the logical extent of the font is inclusive between the Y-coordinate values \((y + \text{font}.\text{ascent})\) and \((y + \text{font}.\text{descent} - 1)\). Typically, the minimum interline spacing between rows of text is given by ascent + descent.

For a character origin at \(x, y\), the bounding box of a character (that is, the smallest rectangle that encloses the character's shape) described in terms of XCharStruct components is a rectangle with its upper-left corner at:

\[
[x + \text{lbearing}, y - \text{ascent}]
\]

Its width is:

\[
\text{rbearing} - \text{lbearing}
\]

Its height is:

\[
\text{ascent} + \text{descent}
\]

The origin for the next character is defined to be:

\[
[x + \text{width}, y]
\]

The lbearing member defines the extent of the left edge of the character ink from the origin. The rbearing member defines the extent of the right edge of the character ink from the origin. The ascent member defines the extent of the top edge of the character ink from the origin. The descent member defines the extent of the bottom edge of the character ink from the origin.
The width member defines the logical width of the character.

Note that the baseline (the y position of the character origin) is logically viewed as being the scanline just below nondescending characters. When descent is zero, only pixels with Y-coordinates less than y are drawn, and the origin is logically viewed as being coincident with the left edge of a nonkerned character. When lbearing is zero, no pixels with X-coordinate less than x are drawn. Any of the XCharStruct metric members could be negative. If the width is negative, the next character will be placed to the left of the current origin.

The X protocol does not define the interpretation of the attributes member in the XCharStruct structure. A nonexistent character is represented with all members of its XCharStruct set to zero.

A font is not guaranteed to have any properties. The interpretation of the property value (for example, long or unsigned long) 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.

**8.5.1. Loading and Freeing Fonts**

Xlib provides functions that you can use to load fonts, get font information, unload fonts, and free font information. A few font functions use a GContext resource ID or a font ID interchangeably.

To load a given font, use XLoadFont.

Font XLoadFont(display, name)

Display *display;
char *name;

display Specifies the connection to the X server.
name Specifies the name of the font, which is a null-terminated string.

The XLoadFont function loads the specified font and returns its associated font ID. If the font name is not in the Host Portable Character Encoding the result is implementation dependent. Use of uppercase or lowercase does not matter. The interpretation of characters "?" and "*" 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. If XLoadFont was unsuccessful at loading the specified font, a BadName error results. Fonts are not associated with a particular screen and can be stored as a component of any GC. When the font is no longer needed, call XUnloadFont.

XLoadFont can generate BadAlloc and BadName errors.

To return information about an available font, use XQueryFont.

XFontStruct XQueryFont(display, font_ID)

Display *display;
XID font_ID;

display Specifies the connection to the X server.
font_ID Specifies the font ID or the GContext ID.

The XQueryFont function returns a pointer to the XFontStruct structure, which contains information associated with the font. You can query a font or the font stored in a GC. The font ID stored in the XFontStruct structure will be the GContext ID, and you need to be careful when using this ID in other functions (see XGContextFromGC). If the font does not exist, XQueryFont returns NULL. To free this data, use XFreeFontInfo.

To perform a XLoadFont and XQueryFont in a single operation, use XLoadQueryFont.
XLoadQueryFont( display, name)

Display *display;
char *name;

display Specifies the connection to the X server.
name Specifies the name of the font, which is a null-terminated string.

The XLoadQueryFont function provides the most common way for accessing a font.
XLoadQueryFont both opens (loads) the specified font and returns a pointer to the appropriate
XFontStruct structure. If the font name is not in the Host Portable Character Encoding
the result is implementation dependent. If the font does not exist, XLoadQueryFont returns
NULL.

XLoadQueryFont can generate a BadAlloc error.

To unload the font and free the storage used by the font structure that was allocated by
XQueryFont or XLoadQueryFont, use XFreeFont.

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

display Specifies the connection to the X server.
font_struct Specifies the storage associated with the font.

The XFreeFont function deletes the association between the font resource ID and the specified
font and frees the XFontStruct structure. The font itself will be freed when no other resource
references it. The data and the font should not be referenced again.

XFreeFont can generate a BadFont error.

To return a given font property, use XGetFontProperty.

Bool XGetFontProperty(font_struct, atom, value_return)
XFontStruct *font_struct;
Atom atom;
unsigned long *value_return;

font_struct Specifies the storage associated with the font.
atom Specifies the atom for the property name you want returned.
value_return Returns the value of the font property.

Given the atom for that property, the XGetFontProperty function returns the value of the
specified font property. XGetFontProperty also returns False if the property was not defined
or True if it was defined. A set of predefined atoms exists for font properties, which can be
found in <X11/Xatom.h>. This set contains the standard properties associated with a font.
Although it is not guaranteed, it is likely that the predefined font properties will be present.

To unload a font that was loaded by XLoadFont, use XUnloadFont.

XUnloadFont( display, font)
Display *display;
Font font;

display Specifies the connection to the X server.
font Specifies the font.

The XUnloadFont function deletes the association between the font resource ID and the
specified font. The font itself will be freed when no other resource references it. The font
should not be referenced again.
XUnloadFont can generate a BadFont error.

8.5.2. Obtaining and Freeing Font Names and Information
You obtain font names and information by matching a wildcard specification when querying a font type for a list of available sizes and so on.

To return a list of the available font names, use XListFonts.

char **XListFonts(display, pattern, maxnames, actual_count_return)
  Display *display;
  char *pattern;
  int maxnames;
  int *actual_count_return;

display Specifies the connection to the X server.
pattern Specifies the null-terminated pattern string that can contain wildcard characters.
maxnames Specifies the maximum number of names to be returned.
actual_count_return Returns the actual number of font names.

The XListFonts function returns an array of available font names (as controlled by the font search path; see XSetFontPath) that match the string you passed to the pattern argument. The pattern string can contain any characters, but each asterisk (*) is a wildcard for any number of characters, and each question mark (?) is a wildcard for a single character. If the pattern string is not in the Host Portable Character Encoding the result is implementation dependent. Use of uppercase or lowercase does not matter. Each returned string is null-terminated. If the data returned by the server is in the Latin Portable Character Encoding, then the returned strings are in the Host Portable Character Encoding. Otherwise, the result is implementation dependent. If there are no matching font names, XListFonts returns NULL. The client should call XFreeFontNames when finished with the result to free the memory.

To free a font name array, use XFreeFontNames.

XFreeFontNames(list)
  char *list[];

list Specifies the array of strings you want to free.

The XFreeFontNames function frees the array and strings returned by XListFonts or XListFontsWithInfo.

To obtain the names and information about available fonts, use XListFontsWithInfo.

char **XListFontsWithInfo(display, pattern, maxnames, count_return, info_return)
  Display *display;
  char *pattern;
  int maxnames;
  int *count_return;
  XFontStruct **info_return;

display Specifies the connection to the X server.
pattern Specifies the null-terminated pattern string that can contain wildcard characters.
maxnames Specifies the maximum number of names to be returned.
count_return Returns the actual number of matched font names.
info_return Returns the font information.
The **XListFontsWithlnfo** function returns a list of font names that match the specified pattern and their associated font information. The list of names is limited to size specified by max-names. The information returned for each font is identical to what **XLoadQueryFont** would return except that the per-character metrics are not returned. The pattern string can contain any characters, but each asterisk (*) is a wildcard for any number of characters, and each question mark (?) is a wildcard for a single character. If the pattern string is not in the Host Portable Character Encoding the result is implementation dependent. Use of uppercase or lowercase does not matter. Each returned string is null-terminated. If the data returned by the server is in the Latin Portable Character Encoding, then the returned strings are in the Host Portable Character Encoding. Otherwise, the result is implementation dependent. If there are no matching font names, **XListFontsWithlnfo** returns NULL.

To free only the allocated name array, the client should call **XFreeFontNames**. To free both the name array and the font information array, or to free just the font information array, the client should call **XFreeFontlnfo**.

To free the the font information array, use **XFreeFontlnfo**.

```c
int XFreeFontlnfo(names, freeinfo, actual_count)
```

- **names** Specifies the list of font names returned by **XListFontsWithlnfo**.
- **freeinfo** Specifies the font information returned by **XListFontsWithlnfo**.
- **actual_count** Specifies the actual number of matched font names returned by **XListFontsWithlnfo**.

The **XFreeFontlnfo** function frees the the font information array. To free an XFontStruct structure without closing the font, call **XFreeFontlnfo** with the names argument specified as NULL.

### 8.5.3. Computing Character String Sizes

Xlib provides functions that you can use to compute the width, the logical extents, and the server information about 8-bit and 2-byte text strings. The width is computed by adding the character widths of all the characters. It does not matter if the font is an 8-bit or 2-byte font. These functions return the sum of the character metrics, in pixels.

To determine the width of an 8-bit character string, use **XTextWidth**.

```c
int XTextWidth(font_struct, string, count)
```

- **font_struct** Specifies the font used for the width computation.
- **string** Specifies the character string.
- **count** Specifies the character count in the specified string.

To determine the width of a 2-byte character string, use **XTextWidthl6**.

```c
int XTextWidthl6(font_struct, string, count)
```

- **font_struct** Specifies the font used for the width computation.
- **string** Specifies the character string.
- **count** Specifies the character count in the specified string.
font_struct  Specifies the font used for the width computation.
string     Specifies the character string.
count      Specifies the character count in the specified string.

8.5.4. Computing Logical Extents
To compute the bounding box of an 8-bit character string in a given font, use XTextExtents.

XTextExtents(font_struct, string, nchars, direction_return, font_ascent_return,
          font_descent_return, overall_return)
XFontStruct *font_struct;
char *string;
int nchars;
int *direction_return;
int *font_ascent_return, *font_descent_return;
XCharStruct *overall_return;

font_struct  Specifies the XFontStruct structure.
string     Specifies the character string.
nchars     Specifies the number of characters in the character string.
direction_return Returns the value of the direction hint (FontLeftToRight or FontRightToLeft).
font_ascent_return  Returns the font ascent.
font_descent_return  Returns the font descent.
overall_return Returns the overall size in the specified XCharStruct structure.

To compute the bounding box of a 2-byte character string in a given font, use XTextExtents16.

XTextExtents16(font_struct, string, nchars, direction_return, font_ascent_return,
          font_descent_return, overall_return)
XFontStruct *font_struct;
XChar2b *string;
int nchars;
int *direction_return;
int *font_ascent_return, *font_descent_return;
XCharStruct *overall_return;

font_struct  Specifies the XFontStruct structure.
string     Specifies the character string.
nchars     Specifies the number of characters in the character string.
direction_return Returns the value of the direction hint (FontLeftToRight or FontRightToLeft).
font_ascent_return  Returns the font ascent.
font_descent_return  Returns the font descent.
overall_return Returns the overall size in the specified XCharStruct structure.
The XTextExtents and XTextExtents16 functions perform the size computation locally and, thereby, avoid the round-trip overhead of XQueryTextExtents and XQueryTextExtents16. Both functions return an XCharStruct structure, whose members are set to the values as follows.

The ascent member is set to the maximum of the ascent metrics of all characters in the string. The descent member is set to the maximum of the descent metrics. The width member is set to the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string. Let L be the left-side-bearing metric of the character plus W. Let R be the right-side-bearing metric of the character plus W. The lbcaring member is set to the minimum L of all characters in the string. The rbearing member is set to the maximum R.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each XChar2b structure is interpreted as a 16-bit number with byte1 as the most-significant byte. If the font has no defined default character, undefined characters in the string are taken to have all zero metrics.

8.5.5. Querying Character String Sizes

To query the server for the bounding box of an 8-bit character string in a given font, use XQueryTextExtents.

XQueryTextExtents(display, font_ID, string, nchars, direction_return, font_ascent_return, font_descent_return, overall_return)

Display *display;
XID font_ID;
char *string;
int nchars;
int *direction_return;
int *font_ascent_return, *font_descent_return;
XCharStruct *overall_return;

display Specifies the connection to the X server.
font_ID Specifies either the font ID or the GContext ID that contains the font.
string Specifies the character string.
nchars Specifies the number of characters in the character string.
direction_return Returns the value of the direction hint (FontLeftToRight or FontRightToLeft).

font_ascent_return
Returns the font ascent.

font_descent_return
Returns the font descent.

overall_return Returns the overall size in the specified XCharStruct structure.

To query the server for the bounding box of a 2-byte character string in a given font, use XQueryTextExtents16.
XQueryTextExtents16(display, font_ID, string, nchars, direction_return, font_ascent_return, font_descent_return, overall_return)
  Display *display;
  XID font_ID;
  XChar2b *string;
  int nchars;
  int *direction_return;
  int *font_ascent_return, *font_descent_return;
  XCharStruct *overall_return;

  display Specifies the connection to the X server.
  font_ID Specifies either the font ID or the GContext ID that contains the font.
  string Specifies the character string.
  nchars Specifies the number of characters in the character string.
  direction_return Returns the value of the direction hint (FontLeftToRight or FontRightToLeft).
  font_ascent_return Returns the font ascent.
  font_descent_return Returns the font descent.
  overall_return Returns the overall size in the specified XCharStruct structure.

The XQueryTextExtents and XQueryTextExtents16 functions return the bounding box of the specified 8-bit and 16-bit character string in the specified font or the font contained in the specified GC. These functions query the X server and, therefore, suffer the round-trip overhead that is avoided by XTextExtents and XTextExtents16. Both functions return a XCharStruct structure, whose members are set to the values as follows.

The ascent member is set to the maximum of the ascent metrics of all characters in the string. The descent member is set to the maximum of the descent metrics. The width member is set to the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string. Let L be the left-side-bearing metric of the character plus W. Let R be the right-side-bearing metric of the character plus W. The lbcaring member is set to the minimum L of all characters in the string. The rbearing member is set to the maximum R.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each XChar2b structure is interpreted as a 16-bit number with byte1 as the most-significant byte. If the font has no defined default character, undefined characters in the string are taken to have all zero metrics.

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

XQueryTextExtents and XQueryTextExtents16 can generate BadFont and BadGC errors.

8.6. Drawing Text
This section discusses how to draw:
- Complex text
- Text characters
- Image text characters

The fundamental text functions XDrawText and XDrawText16 use the following structures.

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

typedef struct {
   XChar2b *chars; /* pointer to two-byte characters */
   int nchars; /* number of characters */
   int delta; /* delta between strings */
   Font font; /* font to print it in, None don’t change */
} XTextItem16;

If the font member is not None, the font is changed before printing and also is stored in the GC. If an error was generated during text drawing, the previous items may have been drawn. The baseline of the characters are drawn starting at the x and y coordinates that you pass in the text drawing functions.

For example, consider the background rectangle drawn by XDrawImageString. If you want the upper-left corner of the background rectangle to be at pixel coordinate (x,y), pass the (x,y + ascent) as the baseline origin coordinates to the text functions. The ascent is the font ascent, as given in the XFontStruct structure. If you want the lower-left corner of the background rectangle to be at pixel coordinate (x,y), pass the (x,y - descent + 1) as the baseline origin coordinates to the text functions. The descent is the font descent, as given in the XFontStruct structure.

8.6.1. Drawing Complex Text

To draw 8-bit characters in a given drawable, use XDrawText.

XDrawText(display, d, gc, x, y, items, nitems)

Display *display;
Drawable d;
GC gc;
int x, y;
XTextItem *items;
int nitems;

display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
x
y Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.
items Specifies an array of text items.
nitems Specifies the number of text items in the array.

To draw 2-byte characters in a given drawable, use XDrawText16.

XDrawText16(display, d, gc, x, y, items, nitems)

Display *display;
Drawable d;
GC gc;
int x, y;
XTextItem16 *items;
int nitems;
display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
x
y Specify the x and y coordinates, which are relative to the origin of the
specified drawable and define the origin of the first character.
items Specifies an array of text items.
nitems Specifies the number of text items in the array.

The XDrawText16 function is similar to XDrawText except that it uses 2-byte or 16-bit
characters. Both functions allow complex spacing and font shifts between counted strings.
Each text item is processed in turn. A font member other than None in an item causes the
font to be stored in the GC and used for subsequent text. A text element delta specifies an
additional change in the position along the x axis before the string is drawn. The delta is
always added to the character origin and is not dependent on any characteristics of the font.
Each character image, as defined by the font in the GC, is treated as an additional mask for a
fill operation on the drawable. The drawable is modified only where the font character has a
bit set to 1. If a text item generates a BadFont error, the previous text items may have been
drawn.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each XChar2b
structure is interpreted as a 16-bit number with byte1 as the most-significant byte.
Both functions use these GC components: function, plane-mask, fill-style, font, subwindow-
mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent
components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XDrawText and XDrawText16 can generate BadDrawable, BadFont, BadGC, and Bad-
Match errors.

8.6.2. Drawing Text Characters
To draw 8-bit characters in a given drawable, use XDrawString.

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

To draw 2-byte characters in a given drawable, use XDrawString16.
XDrawString16(display, d, gc, x, y, string, length)

Display *display;
Drawable d;
GC gc;
int x, y;
XChar2b *string;
int length;

display  Specifies the connection to the X server.
d  Specifies the drawable.
gc  Specifies the GC.
x
y  Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.
string  Specifies the character string.
length  Specifies the number of characters in the string argument.

Each character image, as defined by the font in the GC, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1. For fonts defined with 2-byte matrix indexing and used with XDrawString16, each byte is used as a byte2 with a byte1 of zero.

Both functions use these GC components: function, plane-mask, fill-style, font, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XDrawString and XDrawString16 can generate BadDrawable, BadGC, and BadMatch errors.

8.6.3. Drawing Image Text Characters

Some applications, in particular terminal emulators, need to print image text in which both the foreground and background bits of each character are painted. This prevents annoying flicker on many displays.

To draw 8-bit image text characters in a given drawable, use XDrawImageString.

XDrawImageString(display, d, gc, x, y, string, length)

Display *display;
Drawable d;
GC gc;
int x, y;
char *string;
int length;

display  Specifies the connection to the X server.
d  Specifies the drawable.
gc  Specifies the GC.
x
y  Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.
string  Specifies the character string.
length  Specifies the number of characters in the string argument.

To draw 2-byte image text characters in a given drawable, use XDrawImageString16.
XDrawImageString16(display, d, gc, x, y, string, length)

Display *display;
Drawable d;
GC gc;
int x, y;
XChar2b *string;
int length;

display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
x y Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.
string Specifies the character string.
length Specifies the number of characters in the string argument.

The XDrawImageString16 function is similar to XDrawImageString except that it uses 2-byte or 16-bit characters. Both functions also use both the foreground and background pixels of the GC in the destination.

The effect is first to fill a destination rectangle with the background pixel defined in the GC and then to paint the text with the foreground pixel. The upper-left corner of the filled rectangle is at:

\[ [x, y - \text{font-ascent}] \]

The width is:

overall-width

The height is:

\[ \text{font-ascent} + \text{font-descent} \]

The overall-width, font-ascent, and font-descent are as would be returned by XQueryTextExtents using gc and string. The function and fill-style defined in the GC are ignored for these functions. The effective function is GXcopy, and the effective fill-style is FillSolid.

For fonts defined with 2-byte matrix indexing and used with XDrawImageString, each byte is used as a byte2 with a byte1 of zero.

Both functions use these GC components: plane-mask, foreground, background, font, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask.

XDrawImageString and XDrawImageString16 can generate BadDrawable, BadGC, and BadMatch errors.

8.7. Transferring Images between Client and Server

Xlib provides functions that you can use to transfer images between a client and the server. Because the server may require diverse data formats, Xlib provides an image object that fully describes the data in memory and that provides for basic operations on that data. You should reference the data through the image object rather than referencing the data directly. However, some implementations of the Xlib library may efficiently deal with frequently used data formats by replacing functions in the procedure vector with special case functions. Supported operations include destroying the image, getting a pixel, storing a pixel, extracting a subimage of an image, and adding a constant to an image (see section 16.5).
All the image manipulation functions discussed in this section make use of the XImage data structure, which describes an image as it exists in the client’s memory.

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

You may request that some of the members (for example, height, width, and xoffset) be changed when the image is sent to the server. That is, you may send a subset of the image. Other members (for example, byte_order, bitmap_unit, and so forth) are characteristics of both the image and the server. If these members differ between the image and the server, XPutImage makes the appropriate conversions. The first byte of the first scanline of plane n is located at the address (data + (n * height * bytes_per_line)).

To combine an image in memory with a rectangle of a drawable on the display, use XPutImage.

```
XPutImage(display, d, gc, image, src_x, src_y, dest_x, dest_y, width, height)
    Display *display;
    Drawable d;
    GC gc;
    XImage *image;
    int src_x, src_y;
    int dest_x, dest_y;
    unsigned int width, height;

display          Specifies the connection to the X server.
d               Specifies the drawable.
_gc             Specifies the GC.
_image          Specifies the image you want combined with the rectangle.
src_x            Specifies the offset in X from the left edge of the image defined by the XImage data structure.
```
Specifies the offset in \( Y \) from the top edge of the image defined by the \texttt{XImage} data structure.

Specify the \( x \) and \( y \) coordinates, which are relative to the origin of the drawable and are the coordinates of the subimage.

Specify the width and height of the subimage, which define the dimensions of the rectangle.

The \texttt{XPutImage} function combines an image in memory with a rectangle of the specified drawable. If \texttt{XYBitmap} format is used, the depth of the image must be one, or a \texttt{BadMatch} error results. The foreground pixel in the \texttt{GC} defines the source for the one bits in the image, and the background pixel defines the source for the zero bits. For \texttt{XPixmap} and \texttt{ZPixmap}, the depth of the image must match the depth of the drawable, or a \texttt{BadMatch} error results. The section of the image defined by the \texttt{src\_x,\ src\_y, width, and height} arguments is drawn on the specified part of the drawable.

This function uses these \texttt{GC} components: \texttt{function, plane\_mask, subwindow\_mode, clip\_x\_origin, clip\_y\_origin, and clip\_mask}. It also uses these \texttt{GC} mode-dependent components: \texttt{foreground} and \texttt{background}.

\texttt{XPutImage} can generate \texttt{BadDrawable, BadGC, BadMatch, and BadValue} errors.

To return the contents of a rectangle in a given drawable on the display, use \texttt{XGetImage}.

This function specifically supports rudimentary screen dumps.

\texttt{XImage *XGetImage(display, d, x, y, width, height, plane\_mask, format)}

\begin{itemize}
  \item \texttt{display} specifies the connection to the X server.
  \item \texttt{d} specifies the drawable.
  \item \texttt{x, y} specify the \( x \) and \( y \) coordinates, which are relative to the origin of the drawable and define the upper-left corner of the rectangle.
  \item \texttt{width, height} specify the width and height of the subimage, which define the dimensions of the rectangle.
  \item \texttt{plane\_mask} specifies the plane mask.
  \item \texttt{format} specifies the format for the image. You can pass \texttt{XPixmap} or \texttt{ZPixmap}.
\end{itemize}

The \texttt{XGetImage} function returns a pointer to an \texttt{XImage} structure. This structure provides you with the contents of the specified rectangle of the drawable in the format you specify. If the format argument is \texttt{XPixmap}, the image contains only the bit planes you passed to the plane_mask argument. If the plane_mask argument only requests a subset of the planes of the display, the depth of the returned image will be the number of planes requested. If the format argument is \texttt{ZPixmap}, \texttt{XGetImage} returns as zero the bits in all planes not specified in the plane_mask argument. The function performs no range checking on the values in plane_mask and ignores extraneous bits.

\texttt{XGetImage} returns the depth of the image to the depth member of the \texttt{XImage} structure. The depth of the image is as specified when the drawable was created, except when getting a subset of the planes in \texttt{XPixmap} format, when the depth is given by the number of bits set
to 1 in plane_mask.

If the drawable is a pixmap, the given rectangle must be wholly contained within the pixmap, or a BadMatch 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 BadMatch error results. Note that the borders of the window can be included and read with this request. If the window has backing-store, the backing-store contents are returned for regions of the window that are obscured by noninferior windows. If the window does not have backing-store, the returned contents of such obscured regions are undefined. The returned contents of visible regions of inferiors of a different depth than the specified window's depth are also undefined. The pointer cursor image is not included in the returned contents. If a problem occurs, XGetImage returns NULL.

XGetImage can generate BadDrawable, BadMatch, and BadValue errors.

To copy the contents of a rectangle on the display to a location within a preexisting image structure, use XGetSubImage.

XImage *XGetSubImage(display, d, x, y, width, height, plane_mask, format, dest_image, dest_x, dest_y)

Display *display;
Drawable d;
int x, y;
unsigned int width, height;
unsigned long plane_mask;
int format;
XImage *dest_image;
int dest_x, dest_y;

display Specifies the connection to the X server.
d Specifies the drawable.
x y Specify the x and y coordinates, which are relative to the origin of the drawable and define the upper-left corner of the rectangle.
width height Specify the width and height of the subimage, which define the dimensions of the rectangle.
plane_mask Specifies the plane mask.
format Specifies the format for the image. You can pass XYPixmap or ZPixmap.
dest_image Specify the destination image.
dest_x dest_y Specify the x and y coordinates, which are relative to the origin of the destination rectangle, specify its upper-left corner, and determine where the subimage is placed in the destination image.

The XGetSubImage function updates dest_image with the specified subimage in the same manner as XGetImage. If the format argument is XYPixmap, the image contains only the bit planes you passed to the plane_mask argument. If the format argument is ZPixmap, XGetSubImage returns as zero the bits in all planes not specified in the plane_mask argument. The function performs no range checking on the values in plane_mask and ignores extraneous bits. As a convenience, XGetSubImage returns a pointer to the same XImage structure specified by dest_image.

The depth of the destination XImage structure must be the same as that of the drawable. If the specified subimage does not fit at the specified location on the destination image, the right
and bottom edges are clipped. If the drawable is a pixmap, the given rectangle must be wholly contained within the pixmap, or a BadMatch error results. If the drawable is a window, the window must be drawable, 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 BadMatch error results. If the window has backing-store, then the backing-store contents are returned for regions of the window that are obscured by noninferior windows. If the window does not have backing-store, the returned contents of such obscured regions are undefined. The returned contents of visible regions of inferiors of a different depth than the specified window’s depth are also undefined. If a problem occurs, XGetSubImage returns NULL.

XGetSubImage can generate BadDrawable, BadGC, BadMatch, and BadValue errors.
Chapter 9
Window and Session Manager Functions

Although it is difficult to categorize functions as exclusively for an application or a window manager or a session manager, the functions in this chapter are most often used by window managers and session managers. It is not expected that these functions will be used by most application programs. Xlib provides management functions to:

- Change the parent of a window
- Control the lifetime of a window
- Manage installed colormaps
- Set and retrieve the font search path
- Grab the server
- Kill a client
- Control the screen saver
- Control host access

9.1. Changing the Parent of a Window

To change a window's parent to another window on the same screen, use XReparentWindow. There is no way to move a window between screens.

XReparentWindow(display, w, parent, x, y)

- display Specifies the connection to the X server.
- w Specifies the window.
- parent Specifies the parent window.
- x y Specify the x and y coordinates of the position in the new parent window.

If the specified window is mapped, XReparentWindow automatically performs an UnmapWindow request on it, removes it from its current position in the hierarchy, and inserts it as the child of the specified parent. The window is placed in the stacking order on top with respect to sibling windows.

After re-parenting the specified window, XReparentWindow causes the X server to generate a ReparentNotify event. The override redirect member returned in this event is set to the window's corresponding attribute. Window manager clients usually should ignore this window if this member is set to True. Finally, if the specified window was originally mapped, the X server automatically performs a MapWindow request on it.

The X server performs normal exposure processing on formerly obscured windows. The X server might not generateExpose events for regions from the initial UnmapWindow request that are immediately obscured by the final MapWindow request. A BadMatch error results if:

- The new parent window is not on the same screen as the old parent window.

147
The new parent window is the specified window or an inferior of the specified window.
- The new parent is InputOnly and the window is not.
- The specified window has a ParentRelative background, and the new parent window is not the same depth as the specified window.

XReparentWindow can generate BadMatch and BadWindow errors.

9.2. Controlling the Lifetime of a Window

The save-set of a client is a list of other clients' windows that, if they are inferiors of one of the client's windows at connection close, should not be destroyed and should be remapped if they are unmapped. For further information about close-connection processing, see section 2.6.

To allow an application's window to survive when a window manager that has reparented a window fails, Xlib provides the save-set functions that you can use to control the longevity of subwindows that are normally destroyed when the parent is destroyed. For example, a window manager that wants to add decoration to a window by adding a frame might reparent an application's window. When the frame is destroyed, the application's window should not be destroyed but be returned to its previous place in the window hierarchy.

The X server automatically removes windows from the save-set when they are destroyed.

To add or remove a window from the client's save-set, use XChangeSaveSet.

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

display Specifies the connection to the X server.
w Specifies the window that you want to add to or delete from the client's save-set.
change_mode Specifies the mode. You can pass SetModelnsert or SetModeDelete.

Depending on the specified mode, XChangeSaveSet either inserts or deletes the specified window from the client's save-set. The specified window must have been created by some other client, or a BadMatch error results.

XChangeSaveSet can generate BadMatch, BadValue, and BadWindow errors.

To add a window to the client's save-set, use XAddToSaveSet.

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

display Specifies the connection to the X server.
w Specifies the window that you want to add to the client's save-set.

The XAddToSaveSet function adds the specified window to the client's save-set. The specified window must have been created by some other client, or a BadMatch error results.

XAddToSaveSet can generate BadMatch and BadWindow errors.

To remove a window from the client's save-set, use XRemoveFromSaveSet.

XRemoveFromSaveSet(display, w)
- Display *display;
- Window w;
display Specifies the connection to the X server.
w Specifies the window that you want to delete from the client's save-set.

The XRemoveFromSaveSet function removes the specified window from the client's save-set. The specified window must have been created by some other client, or a BadMatch error results.

XRemoveFromSaveSet can generate BadMatch and BadWindow errors.

9.3. Managing Installed Colormaps

The X server maintains a list of installed colormaps. Windows using these colormaps are guaranteed to display with correct colors; windows using other colormaps may or may not display with correct colors. Xlib provides functions that you can use to install a colormap, uninstall a colormap, and obtain a list of installed colormaps.

At any time, there is a subset of the installed maps that is viewed as an ordered list and is called the required list. The length of the required list is at most M, where M is the minimum number of installed colormaps specified for the screen in the connection setup. The required list is maintained as follows. When a colormap is specified to XInstallColormap, it is added to the head of the list; the list is truncated at the tail, if necessary, to keep its length to at most M. When a colormap is specified to XUninstallColormap 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 implicitly installed by the X server, and the X server cannot implicitly uninstall a colormap that is in the required list.

To install a colormap, use XInstallColormap.

XInstallColormap(display, colormap)
   Display *display;
   Colormap colormap;

display Specifies the connection to the X server.
colormap Specifies the colormap.

The XInstallColormap function installs the specified colormap for its associated screen. All windows associated with this colormap immediately display with true colors. You associated the windows with this colormap when you created them by calling XCreateWindow, XCreateSimpleWindow, XChangeWindowAttributes, or XSetWindowColormap.

If the specified colormap is not already an installed colormap, the X server generates a colormapNotify event on each window that has that colormap. In addition, for every other colormap that is installed as a result of a call to XInstallColormap, the X server generates a colormapNotify event on each window that has that colormap.

XInstallColormap can generate a BadColor error.

To uninstall a colormap, use XUninstallColormap.

XUninstallColormap(display, colormap)
   Display *display;
   Colormap colormap;

display Specifies the connection to the X server.
colormap Specifies the colormap.

The XUninstallColormap function removes the specified colormap from the required list for its screen. As a result, the specified colormap might be uninstalled, and the X server might implicitly install or uninstall additional colormaps. Which colormaps get installed or uninstalled is server-dependent except that the required list must remain installed.
If the specified colormap becomes uninstalled, the X server generates a \texttt{ColormapNotify} event on each window that has that colormap. In addition, for every other colormap that is installed or uninstalled as a result of a call to \texttt{XUninstallColormap}, the X server generates a \texttt{ColormapNotify} event on each window that has that colormap.

\texttt{XUninstallColormap} can generate a \texttt{BadColor} error.

To obtain a list of the currently installed colormaps for a given screen, use \texttt{XListInstalledColormaps}.

\begin{verbatim}
Colormap * XListInstalledColormaps (display, w, num_return)
  Display *display;
  Window w;
  int *num_return;

display Specifies the connection to the X server.

w Specifies the window that determines the screen.

num_return Returns the number of currently installed colormaps.
\end{verbatim}

The \texttt{XListInstalledColormaps} function returns a list of the currently installed colormaps for the screen of the specified window. The order of the colormaps in the list is not significant and is no explicit indication of the required list. When the allocated list is no longer needed, free it by using \texttt{XFree}.

\texttt{XListInstalledColormaps} can generate a \texttt{BadWindow} error.

### 9.4. Setting and Retrieving the Font Search Path

The set of fonts available from a server depends on a font search path. Xlib provides functions to set and retrieve the search path for a server.

To set the font search path, use \texttt{XSetFontPath}.

\begin{verbatim}
XSetFontPath (display, directories, ndirs)
  Display *display;
  char **directories;
  int ndirs;

display Specifies the connection to the X server.

directories Specifies the directory path used to look for a font. Setting the path to the empty list restores the default path defined for the X server.

ndirs Specifies the number of directories in the path.
\end{verbatim}

The \texttt{XSetFontPath} function defines the directory search path for font lookup. There is only one search path per X server, not one per client. The encoding and interpretation of the strings is implementation dependent, but typically they specify directories or font servers to be searched in the order listed. An X server is permitted to cache font information internally, for example, it might cache an entire font from a file and not check on subsequent opens of that font to see if the underlying font file has changed. However, when the font path is changed the X server is guaranteed to flush all cached information about fonts for which there currently are no explicit resource IDs allocated. The meaning of an error from this request is implementation dependent.

\texttt{XSetFontPath} can generate a \texttt{BadValue} error.

To get the current font search path, use \texttt{XGetFontPath}.

To the specified colormap becomes uninstalled, the X server generates a \texttt{ColormapNotify} event on each window that has that colormap. In addition, for every other colormap that is installed or uninstalled as a result of a call to \texttt{XUninstallColormap}, the X server generates a \texttt{ColormapNotify} event on each window that has that colormap.

\texttt{XUninstallColormap} can generate a \texttt{BadColor} error.
char **XGetFontPath(display, npaths_return)
    Display *display;
    int *npaths_return;

display Specifies the connection to the X server.
npaths_return Returns the number of strings in the font path array.
The XGetFontPath function allocates and returns an array of strings containing the search path. The contents of these strings are implementation dependent and are not intended to be interpreted by client applications. When it is no longer needed, the data in the font path should be freed by using XFreeFontPath.

To free data returned by XGetFontPath, use XFreeFontPath.
XFreeFontPath(list)
    char **list;

list Specifies the array of strings you want to free.
The XFreeFontPath function frees the data allocated by XGetFontPath.

9.5. Server Grabbing
Xlib provides functions that you can use to grab and ungrab the server. These functions can be used to control processing of output on other connections by the window system server. While the server is grabbed, no processing of requests or close downs on any other connection will occur. A client closing its connection automatically ungrabs the server. Although grabbing the server is highly discouraged, it is sometimes necessary.

To grab the server, use XGrabServer.
XGrabServer(display)
    Display *display;

display Specifies the connection to the X server.
The XGrabServer function disables processing of requests and close downs on all other connections than the one this request arrived on. You should not grab the X server any more than is absolutely necessary.

To ungrab the server, use XUngrabServer.
XUngrabServer(display)
    Display *display;

display Specifies the connection to the X server.
The XUngrabServer function restarts processing of requests and close downs on other connections. You should avoid grabbing the X server as much as possible.

9.6. Killing Clients
Xlib provides a function to cause the connection to a client to be closed and its resources to be destroyed. To destroy a client, use XKillClient.
XKillClient(display, resource)
    Display *display;
    XID resource;
display Specifies the connection to the X server.
resource Specifies any resource associated with the client that you want to destroy or AllTemporary.

The XKillClient function forces a close-down of the client that created the resource if a valid resource is specified. If the client has already terminated in either RetainPermanent or RetainTemporary mode, all of the client’s resources are destroyed. If AllTemporary is specified, the resources of all clients that have terminated in RetainTemporary are destroyed (see section 2.5). This permits implementation of window manager facilities that aid debugging. A client can set its close-down mode to RetainTemporary. If the client then crashes, its windows would not be destroyed. The programmer can then inspect the application’s window tree and use the window manager to destroy the zombie windows.

XKillClient can generate a BadValue error.

9.7. Screen Saver Control
Xlib provides functions that you can use to set or reset the mode of the screen saver, to force or activate the screen saver, or to obtain the current screen saver values.

To set the screen saver mode, use XSetScreenSaver.

XSetScreenSaver(display, timeout, interval, prefer_blanking, allow_exposures)
  Display *display;
  int timeout, interval;
  int prefer_blanking;
  int allow_exposures;

display Specifies the connection to the X server.
timeout Specifies the timeout, in seconds, until the screen saver turns on.
interval Specifies the interval, in seconds, between screen saver alterations.
prefer_blanking Specifies how to enable screen blanking. You can pass DontPreferBlanking, PreferBlanking, or DefaultBlanking.
allow_exposures Specifies the screen save control values. You can pass DontAllowExposures, AllowExposures, or DefaultExposures.

Timeout and interval are specified in seconds. A timeout of 0 disables the screen saver (but an activated screen saver is not deactivated), and a timeout of -1 restores the default. Other negative values generate a BadValue error. If the timeout value is nonzero, XSetScreenSaver enables the screen saver. An interval of 0 disables the random-pattern motion. If no input from devices (keyboard, mouse, and so on) is generated for the specified number of timeout seconds once the screen saver is enabled, the screen saver is activated.

For each screen, if blanking is preferred and the hardware supports video blanking, the screen simply goes blank. Otherwise, if either exposures are allowed or the screen can be regenerated without sending Expose events to clients, the screen is tiled with the root window background tile randomly re-originated each interval minutes. Otherwise, the screen’s state do not change, and the screen saver is not activated. The screen saver is deactivated, and all screen states are restored at the next keyboard or pointer input or at the next call to XForceScreenSaver with mode ScreenSaverReset.

If the server-dependent screen saver method supports periodic change, the interval argument serves as a hint about how long the change period should be, and zero hints that no periodic change should be made. Examples of ways to change the screen include scrambling the colormap periodically, moving an icon image around the screen periodically, or tiling the screen with the root window background tile, randomly re-originated periodically.

XSetScreenSaver can generate a BadValue error.
To force the screen saver on or off, use XForceScreenSaver.

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

    display Specifies the connection to the X server.
    mode Specifies the mode that is to be applied. You can pass ScreenSaverActive or ScreenSaverReset.

If the specified mode is ScreenSaverActive and the screen saver currently is deactivated,
XForceScreenSaver activates the screen saver even if the screen saver had been disabled with
a timeout of zero. If the specified mode is ScreenSaverReset and the screen saver currently
is enabled, XForceScreenSaver deactivates the screen saver if it was activated, and the activation
timer is reset to its initial state (as if device input had been received).

XForceScreenSaver can generate a BadValue error.

To activate the screen saver, use XActivateScreenSaver.

XActivateScreenSaver(display)
    Display *display;

To reset the screen saver, use XResetScreenSaver.

XResetScreenSaver(display)
    Display *display;

To get the current screen saver values, use XGetScreenSaver.

XGetScreenSaver(display, timeout_return, interval_return, prefer_blanking_return,
    allow_exposures_return)
    Display *display;
    int *timeout_return, *interval_return;
    int *prefer_blanking_return;
    int *allow_exposures_return;

    display Specifies the connection to the X server.
    timeout_return Returns the timeout, in seconds, until the screen saver turns on.
    interval_return Returns the interval between screen saver invocations.
    prefer_blanking_return
        Returns the current screen blanking preference (DontPreferBlanking, PreferBlanking, or DefaultBlanking).
    allow_exposures_return
        Returns the current screen save control value (DontAllowExposures, AllowExposures, or DefaultExposures).

9.8. Controlling Host Access
This section discusses how to:
- Add, get, or remove hosts from the access control list
- Change, enable, or disable access

X does not provide any protection on a per-window basis. If you find out the resource ID of a
resource, you can manipulate it. To provide some minimal level of protection, however,
connections are permitted only from machines you trust. This is adequate on single-user workstations but obviously breaks down on timesharing machines. Although provisions exist in the X protocol for proper connection authentication, the lack of a standard authentication server leaves host-level access control as the only common mechanism.

The initial set of hosts allowed to open connections typically consists of:

- The host the window system is running on.
- On POSIX-conformant systems, each host listed in the /etc/X?.hosts file. The ? indicates the number of the display. This file should consist of host names separated by newlines. DECnet nodes must terminate in :: to distinguish them from Internet hosts.

If a host is not in the access control list when the access control mechanism is enabled and if the host attempts to establish a connection, the server refuses the connection. To change the access list, the client must reside on the same host as the server and/or must have been granted permission in the initial authorization at connection setup.

Servers also can implement other access control policies in addition to or in place of this host access facility. For further information about other access control implementations, see "X Window System Protocol."

9.8.1. Adding, Getting, or Removing Hosts

Xlib provides functions that you can use to add, get, or remove hosts from the access control list. All the host access control functions use the XHostAddress structure, which contains:

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

The family member specifies which protocol address family to use (for example, TCP/IP or DECnet) and can be FamilyInternet, FamilyDECnet, or FamilyChaos. The length member specifies the length of the address in bytes. The address member specifies a pointer to the address.

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

To add a single host, use XAddHost.

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

`display` Specifies the connection to the X server.
`host` Specifies the host that is to be added.

The XAddHost function adds the specified host to the access control list for that display. The server must be on the same host as the client issuing the command, or a BadAccess error results.

XAddHost can generate BadAccess and BadValue errors.

To add multiple hosts at one time, use XAddHosts.
XAddHosts(display, hosts, num_hosts)
  Display *display;
  XHostAddress *hosts;
  int num_hosts;

display Specifications the connection to the X server.
hosts Specifies each host that is to be added.
num_hosts Specifies the number of hosts.

The XAddHosts function adds each specified host to the access control list for that display.
The server must be on the same host as the client issuing the command, or a BadAccess error
results.

XAddHosts can generate BadAccess and BadValue errors.

To obtain a host list, use XListHosts.

XHostAddress *XListHosts(display, nhosts_return, state_return)
  Display *display;
  int *nhosts_return;
  Bool *state_return;

display Specifies the connection to the X server.
nhosts_return Returns the number of hosts currently in the access control list.
state_return Returns the state of the access control.

The XListHosts function returns the current access control list as well as whether the use of
the list at connection setup was enabled or disabled. XListHosts allows a program to find out
what machines can make connections. It also returns a pointer to a list of host structures that
were allocated by the function. When no longer needed, this memory should be freed by cal¬
calling XFree.

To remove a single host, use XRemoveHost.

XRemoveHost(display, host)
  Display *display;
  XHostAddress *host;

display Specifies the connection to the X server.
host Specifies the host that is to be removed.

The XRemoveHost function removes the specified host from the access control list for that
display. The server must be on the same host as the client process, or a BadAccess error
results. If you remove your machine from the access list, you can no longer connect to that
server, and this operation cannot be reversed unless you reset the server.

XRemoveHost can generate BadAccess and BadValue errors.

To remove multiple hosts at one time, use XRemoveHosts.

XRemoveHosts(display, hosts, num_hosts)
  Display *display;
  XHostAddress *hosts;
  int num_hosts;

display Specifies the connection to the X server.
hosts Specifies each host that is to be removed.
num_hosts Specifies the number of hosts.
The XRemoveHosts function removes each specified host from the access control list for that display. The X server must be on the same host as the client process, or a BadAccess error results. If you remove your machine from the access list, you can no longer connect to that server, and this operation cannot be reversed unless you reset the server.

XRemoveHosts can generate BadAccess and BadValue errors.

9.8.2. Changing, Enabling, or Disabling Access Control

Xlib provides functions that you can use to enable, disable, or change access control.

For these functions to execute successfully, the client application must reside on the same host as the X server and/or have been given permission in the initial authorization at connection setup.

To change access control, use XSetAccessControl.

XSetAccessControl(display, mode)

Display *display;
int mode;

display Specifies the connection to the X server.
mode Specifies the mode. You can pass EnableAccess or DisableAccess.

The XSetAccessControl function either enables or disables the use of the access control list at each connection setup.

XSetAccessControl can generate BadAccess and BadValue errors.

To enable access control, use XEnableAccessControl.

XEnableAccessControl(display)

Display *display;

display Specifies the connection to the X server.

The XEnableAccessControl function enables the use of the access control list at each connection setup.

XEnableAccessControl can generate a BadAccess error.

To disable access control, use XDisableAccessControl.

XDisableAccessControl(display)

Display *display;

display Specifies the connection to the X server.

The XDisableAccessControl function disables the use of the access control list at each connection setup.

XDisableAccessControl can generate a BadAccess error.
Chapter 10

Events

A client application communicates with the X server through the connection you establish with the XOpenDisplay function. A client application sends requests to the X server over this connection. These requests are made by the Xlib functions that are called in the client application. Many Xlib functions cause the X server to generate events, and the user's typing or moving the pointer can generate events asynchronously. The X server returns events to the client on the same connection.

This chapter discusses the following topics associated with events:

- Event types
- Event structures
- Event mask
- Event processing

Functions for handling events are dealt with in the next chapter.

10.1. Event Types

An event is data generated asynchronously by the X server as a result of some device activity or as side effects of a request sent by an Xlib function. Device-related events propagate from the source window to ancestor windows until some client application has selected that event type or until the event is explicitly discarded. The X server generally sends an event to a client application only if the client has specifically asked to be informed of that event type, typically by setting the event-mask attribute of the window. The mask can also be set when you create a window or by changing the window's event-mask. You can also mask out events that would propagate to ancestor windows by manipulating the do-not-propagate mask of the window's attributes. However, MappingNotify events are always sent to all clients.

An event type describes a specific event generated by the X server. For each event type, a corresponding constant name is defined in <X11/X.h>, which is used when referring to an event type. The following table lists the event category and its associated event type or types. The processing associated with these events is discussed in section 10.5.

<table>
<thead>
<tr>
<th>Event Category</th>
<th>Event Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard events</td>
<td>KeyPress, KeyRelease</td>
</tr>
<tr>
<td>Pointer events</td>
<td>ButtonPress, ButtonRelease, MotionNotify</td>
</tr>
<tr>
<td>Window crossing events</td>
<td>EnterNotify, LeaveNotify</td>
</tr>
<tr>
<td>Input focus events</td>
<td>FocusIn, FocusOut</td>
</tr>
<tr>
<td>Keymap state notification event</td>
<td>KeymapNotify</td>
</tr>
<tr>
<td>Exposure events</td>
<td>Expose, GraphicsExpose, NoExpose</td>
</tr>
<tr>
<td>Structure control events</td>
<td>CirculateRequest, ConfigureRequest, MapRequest, ResizeRequest</td>
</tr>
</tbody>
</table>
10.2. Event Structures

For each event type, a corresponding structure is declared in `<X11/Xlib.h>`. All the event structures have the following common members:

```c
typedef struct {
  int type;
  unsigned long serial; /* # of last request processed by server */
  Bool send_event; /* true if this came from a SendEvent request */
  Display *display; /* Display the event was read from */
  Window window;
} XAnyEvent;
```

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a GraphicsExpose event to a client application, it sends an `XGraphicsExposeEvent` structure with the type member set to `GraphicsExpose`. The display member is set to a pointer to the display the event was read on. The send_event member is set to `True` if the event came from a `SendEvent` protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The X server can send events at any time in the input stream. Xlib stores any events received while waiting for a reply in an event queue for later use. Xlib also provides functions that allow you to check events in the event queue (see section 11.3).

In addition to the individual structures declared for each event type, the `XEvent` structure is a union of the individual structures declared for each event type. Depending on the type, you should access members of each event by using the `XEvent` union.

```c
typedef union _XEvent {
  int type; /* must not be changed */
  XAnyEvent xany;
  XKeyEvent xkey;
  XButtonEvent xbutton;
  XMotionEvent xmotion;
  XCrossingEvent xcrossing;
  XFocusChangeEvent xfocus;
  XExposeEvent xexpose;
  XGraphicsExposeEvent xgraphicsexpose;
  XNoExposeEvent xnoexpose;
  XVisibilityEvent xvisibility;
  XCreateWindowEvent xcreatewindow;
  XDestroyWindowEvent xdestroywindow;
  XUnmapEvent xunmap;
} XEvent;
```
XMapEvent xmap;
XMapRequestEvent xmaprequest;
XReparentEvent xreparent;
XConfigureEvent xconfigure;
XGravityEvent xgravity;
XResizeRequestEvent xresizerequest;
XConfigureRequestEvent xconfigurerequest;
XCirculateEvent xcirculate;
XCirculateRequestEvent xcirculaterequest;
XPropertyEvent xproperty;
XSelectionClearEvent xselectionclear;
XSelectionRequestEvent xselectionrequest;
XSelectionEvent xselection;
XColormapEvent xcolormap;
XClientMessageEvent xclient;
XMappingEvent xmapping;
XErrorEvent xerror;
XKeymapEvent xkeymap;
long pad[24];
}

XEvent;

An XEvent structure’s first entry always is the type member, which is set to the event type. The second member always is the serial number of the protocol request that generated the event. The third member always is send_event, which is a Bool that indicates if the event was sent by a different client. The fourth member always is a display, which is the display that the event was read from. Except for keymap events, the fifth member always is a window, which has been carefully selected to be useful to toolkit dispatchers. To avoid breaking toolkits, the order of these first five entries is not to change. Most events also contain a time member, which is the time at which an event occurred. In addition, a pointer to the generic event must be cast before it is used to access any other information in the structure.

10.3. Event Masks

Clients select event reporting of most events relative to a window. To do this, pass an event mask to an Xlib event-handling function that takes an event_mask argument. The bits of the event mask are defined in <X11/X.h>. Each bit in the event mask maps to an event mask name, which describes the event or events you want the X server to return to a client application.

Unless the client has specifically asked for them, most events are not reported to clients when they are generated. Unless the client suppresses them by setting graphics-exposures in the GC to False, GraphicsExpose and NoExpose are reported by default as a result of XCopyPlane and XCopyArea. SelectionClear, SelectionRequest, SelectionNotify, or ClientMessage cannot be masked. Selection related events are only sent to clients cooperating with selections (see section 4.5). When the keyboard or pointer mapping is changed, MappingNotify is always sent to clients.

The following table lists the event mask constants you can pass to the event_mask argument and the circumstances in which you would want to specify the event mask:

<table>
<thead>
<tr>
<th>Event Mask</th>
<th>Circumstances</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoEventMask</td>
<td>No events wanted</td>
</tr>
<tr>
<td>KeyPressMask</td>
<td>Keyboard down events wanted</td>
</tr>
<tr>
<td>KeyReleaseMask</td>
<td>Keyboard up events wanted</td>
</tr>
</tbody>
</table>
10.4. Event Processing Overview

The event reported to a client application during event processing depends on which event masks you provide as the event-mask attribute for a window. For some event masks, there is a one-to-one correspondence between the event mask constant and the event type constant. For example, if you pass the event mask ButtonPressMask, the X server sends back only ButtonPress events. Most events contain a time member, which is the time at which an event occurred.

In other cases, one event mask constant can map to several event type constants. For example, if you pass the event mask SubstructureNotifyMask, the X server can send back CirculateNotify, ConfigureNotify, CreateNotify, DestroyNotify, GravityNotify, MapNotify, ReparentNotify, or UnmapNotify events.

In another case, two event masks can map to one event type. For example, if you pass either PointerMotionMask or ButtonMotionMask, the X server sends back a MotionNotify event.

The following table lists the event mask, its associated event type or types, and the structure name associated with the event type. Some of these structures actually are typedefs to a generic structure that is shared between two event types. Note that N.A. appears in columns for which the information is not applicable.

<table>
<thead>
<tr>
<th>Event Mask</th>
<th>Event Type</th>
<th>Structure</th>
<th>Generic Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>ButtonMotionMask</td>
<td>MotionNotify</td>
<td>XPointerMovedEvent</td>
<td>XMotionEvent</td>
</tr>
<tr>
<td>Button1MotionMask</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Button2MotionMask</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event Mask</td>
<td>Event Type</td>
<td>Structure</td>
<td>Generic Structure</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------</td>
<td>----------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Button3MotionMask</td>
<td>ButtonPress</td>
<td>XButtonPressedEvent</td>
<td>XButtonEvent</td>
</tr>
<tr>
<td>Button4MotionMask</td>
<td>ButtonRelease</td>
<td>XButtonReleasedEvent</td>
<td>XButtonEvent</td>
</tr>
<tr>
<td>Button5MotionMask</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ButtonPressMask</td>
<td>ButtonPress</td>
<td>XButtonPressedEvent</td>
<td>XButtonEvent</td>
</tr>
<tr>
<td>ButtonReleaseMask</td>
<td>ButtonRelease</td>
<td>XButtonReleasedEvent</td>
<td>XButtonEvent</td>
</tr>
<tr>
<td>ColormapChangeMask</td>
<td>ColormapNotify</td>
<td>XColormapEvent</td>
<td></td>
</tr>
<tr>
<td>EnterWindowMask</td>
<td>EnterNotify</td>
<td>XEnterWindowEvent</td>
<td>XCrossingEvent</td>
</tr>
<tr>
<td>LeaveWindowMask</td>
<td>LeaveNotify</td>
<td>XLeaveWindowEvent</td>
<td>XCrossingEvent</td>
</tr>
<tr>
<td>ExposureMask</td>
<td>Expose</td>
<td>XExposeEvent</td>
<td></td>
</tr>
<tr>
<td>GCGraphicsExposures in GC</td>
<td>GraphicsNotify</td>
<td>XGraphicsExposeEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NoExpose</td>
<td>XNoExposeEvent</td>
<td></td>
</tr>
<tr>
<td>FocusChangeMask</td>
<td>FocusIn</td>
<td>XFocusInEvent</td>
<td>XFocusChangeEvent</td>
</tr>
<tr>
<td></td>
<td>FocusOut</td>
<td>XFocusOutEvent</td>
<td>XFocusChangeEvent</td>
</tr>
<tr>
<td>KeymapStateMask</td>
<td>KeymapNotify</td>
<td>XKKeymapEvent</td>
<td></td>
</tr>
<tr>
<td>KeyPressMask</td>
<td>KeyPress</td>
<td>XKKeyPressEvent</td>
<td>XKeyPressEvent</td>
</tr>
<tr>
<td>KeyReleaseMask</td>
<td>KeyRelease</td>
<td>XKKeyReleaseEvent</td>
<td>XKeyPressEvent</td>
</tr>
<tr>
<td>OwnerGrabButtonMask</td>
<td>N.A.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PointerMotionMask</td>
<td>MotionNotify</td>
<td>XPointerMovedEvent</td>
<td>XMotionEvent</td>
</tr>
<tr>
<td>PointerMotionHintMask</td>
<td>N.A.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PropertyChangeMask</td>
<td>PropertyNotify</td>
<td>XPropertyEvent</td>
<td></td>
</tr>
<tr>
<td>ResizeRedirectMask</td>
<td>ResizeRequest</td>
<td>XResizeRequestEvent</td>
<td></td>
</tr>
<tr>
<td>StructureNotifyMask</td>
<td>CirculateNotify</td>
<td>XCirculateEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ConfigureNotify</td>
<td>XConfigureEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DestroyNotify</td>
<td>XDestroyWindowEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GravityNotify</td>
<td>XGravityEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MapNotify</td>
<td>XMapEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReparentNotify</td>
<td>XRparentEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UnmapNotify</td>
<td>XUnmapEvent</td>
<td></td>
</tr>
<tr>
<td>SubstructureNotifyMask</td>
<td>CirculateNotify</td>
<td>XCirculateEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ConfigureNotify</td>
<td>XConfigureEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CreateNotify</td>
<td>XCreateWindowEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DestroyNotify</td>
<td>XDestroyWindowEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GravityNotify</td>
<td>XGravityEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MapNotify</td>
<td>XMapEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReparentNotify</td>
<td>XRparentEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UnmapNotify</td>
<td>XUnmapEvent</td>
<td></td>
</tr>
<tr>
<td>SubstructureRedirectMask</td>
<td>CirculateRequest</td>
<td>XCirculateRequestEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ConfigureRequest</td>
<td>XConfigureRequestEvent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MapRequest</td>
<td>XMapRequestEvent</td>
<td></td>
</tr>
<tr>
<td>N.A.</td>
<td>ClientMessage</td>
<td>XClientMessageEvent</td>
<td></td>
</tr>
<tr>
<td>N.A.</td>
<td>MappingNotify</td>
<td>XMappingEvent</td>
<td></td>
</tr>
<tr>
<td>N.A.</td>
<td>SelectionClear</td>
<td>XSelectionClearEvent</td>
<td></td>
</tr>
</tbody>
</table>

161
The sections that follow describe the processing that occurs when you select the different event masks. The sections are organized according to these processing categories:

- Keyboard and pointer events
- Window crossing events
- Input focus events
- Keymap state notification events
- Exposure events
- Window state notification events
- Structure control events
- Colormap state notification events
- Client communication events

10.5. Keyboard and Pointer Events
This section discusses:

- Pointer button events
- Keyboard and pointer events

10.5.1. Pointer Button Events
The following describes the event processing that occurs when a pointer button press is processed with the pointer in some window w and when no active pointer grab is in progress. The X server searches the ancestors of w from the root down, looking for a passive grab to activate. If no matching passive grab on the button exists, the X server automatically starts an active grab for the client receiving the event and sets the last-pointer-grab time to the current server time. The effect is essentially equivalent to an `XGrabButton` with these client passed arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>The event window</td>
</tr>
<tr>
<td>event_mask</td>
<td>The client's selected pointer events on the event window</td>
</tr>
<tr>
<td>pointer_mode</td>
<td>GrabModeAsync</td>
</tr>
<tr>
<td>keyboard_mode</td>
<td>GrabModeAsync</td>
</tr>
<tr>
<td>owner_events</td>
<td>True, if the client has selected OwnerGrabButton-Mask on the event window, otherwise False</td>
</tr>
<tr>
<td>confine_to</td>
<td>None</td>
</tr>
<tr>
<td>cursor</td>
<td>None</td>
</tr>
</tbody>
</table>

The active grab is automatically terminated when the logical state of the pointer has all buttons released. Clients can modify the active grab by calling `XUngrabPointer` and `XChangeActivePointerGrab`.
10.5.2. Keyboard and Pointer Events

This section discusses the processing that occurs for the keyboard events KeyPress and KeyRelease and the pointer events ButtonPress, ButtonRelease, and MotionNotify. For information about the keyboard event-handling utilities, see chapter 11.

The X server reports KeyPress or KeyRelease events to clients wanting information about keys that logically change state. Note that these events are generated for all keys, even those mapped to modifier bits. The X server reports ButtonPress or ButtonRelease events to clients wanting information about buttons that logically change state.

The X server reports MotionNotify events to clients wanting information about when the pointer logically moves. The X server generates this event whenever the pointer is moved and the pointer motion begins and ends in the window. The granularity of MotionNotify events is not guaranteed, but a client that selects this event type is guaranteed to receive at least one event when the pointer moves and then rests.

The generation of the logical changes lags the physical changes if device event processing is frozen.

To receive KeyPress, KeyRelease, ButtonPress, and ButtonRelease events, set KeyPressMask, KeyReleaseMask, ButtonPressMask, and ButtonReleaseMask bits in the event-mask attribute of the window.

To receive MotionNotify events, set one or more of the following event masks bits in the event-mask attribute of the window.

- **Button1MotionMask – Button5MotionMask**
  The client application receives MotionNotify events only when one or more of the specified buttons is pressed.

- **ButtonMotionMask**
  The client application receives MotionNotify events only when at least one button is pressed.

- **PointerMotionMask**
  The client application receives MotionNotify events independent of the state of the pointer buttons.

- **PointerMotionHintMask**
  If PointerMotionHintMask is selected in combination with one or more of the above masks, the X server is free to send only one MotionNotify event (with the is_hint member of the XPointerMovedEvent structure set to NotifyHint) to the client for the event window, until either the key or button state changes, the pointer leaves the event window, or the client calls XQueryPointer or XGetMotionEvents. The server still may send MotionNotify events without is_hint set to NotifyHint.

The source of the event is the viewable window that the pointer is in. The window used by the X server to report these events depends on the window’s position in the window hierarchy and whether any intervening window prohibits the generation of these events. Starting with the source window, the X server searches up the window hierarchy until it locates the first window specified by a client as having an interest in these events. If one of the intervening windows has its do-not-propagate-mask set to prohibit generation of the event type, the events of those types will be suppressed. Clients can modify the actual window used for reporting by performing active grabs and, in the case of keyboard events, by using the focus window.

The structures for these event types contain:

```c
typedef struct {
    int type;            /* ButtonPress or ButtonRelease */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event;     /* true if this came from a SendEvent request */
    Display *display;    /* Display the event was read from */
} XEvent;
```
These structures have the following common members: window, root, subwindow, time, x, y, x_root, y_root, state, and same_screen. The window member is set to the window on which the event was generated and is referred to as the event window. As long as the conditions previously discussed are met, this is the window used by the X server to report the event. The root member is set to the source window’s root window. The x_root and y_root members are set to the pointer’s coordinates relative to the root window’s origin at the time of the event.
The same_screen member is set to indicate whether the event window is on the same screen as the root window and can be either True or False. If True, the event and root windows are on the same screen. If False, the event and root windows are not on the same screen.

If the source window is an inferior of the event window, the subwindow member of the structure is set to the child of the event window that is the source window or the child of the event window that is an ancestor of the source window. Otherwise, the X server sets the subwindow member to None. The time member is set to the time when the event was generated and is expressed in milliseconds.

If the event window is on the same screen as the root window, the x and y members are set to the coordinates relative to the event window’s origin. Otherwise, these members are set to zero.

The state member is set to indicate the logical state of the pointer buttons and modifier keys just prior to the event, which is the bitwise inclusive OR of one or more of the button or modifier key masks: Button1Mask, Button2Mask, Button3Mask, Button4Mask, Button5Mask, ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, and Mod5Mask.

Each of these structures also has a member that indicates the detail. For the XKeyPressedEvent and XKeyReleasedEvent structures, this member is called keycode. It is set to a number that represents a physical key on the keyboard. The keycode is an arbitrary representation for any key on the keyboard (see sections 12.7 and 16.1).

For the XButtonPressedEvent and XButtonReleasedEvent structures, this member is called button. It represents the pointer button that changed state and can be the Button1, Button2, Button3, Button4, or Button5 value. For the XPointerMovedEvent structure, this member is called is_hint. It can be set to NotifyNormal or NotifyHint.

10.6. Window Entry/Exit Events

This section describes the processing that occurs for the window crossing events EnterNotify and LeaveNotify. If a pointer motion or a window hierarchy change causes the pointer to be in a different window than before, the X server reports EnterNotify or LeaveNotify events to clients who have selected for these events. All EnterNotify and LeaveNotify events caused by a hierarchy change are generated after any hierarchy event (UnmapNotify, MapNotify, ConfigureNotify, GravityNotify, CirculateNotify) caused by that change; however, the X protocol does not constrain the ordering of EnterNotify and LeaveNotify events with respect to FocusOut, VisibilityNotify, and Expose events.

This contrasts with MotionNotify events, which are also generated when the pointer moves but only when the pointer motion begins and ends in a single window. An EnterNotify or LeaveNotify event also can be generated when some client application calls XGrabPointer and XUngrabPointer.

To receive EnterNotify or LeaveNotify events, set the EnterWindowMask or LeaveWindowMask bits of the event-mask attribute of the window.

The structure for these event types contains:

typedef struct {
    int type; /* EnterNotify or LeaveNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window window; /* "event" window reported relative to */
    Window root; /* root window that the event occurred on */
    Window subwindow; /* child window */
    Time time; /* milliseconds */
    int x, y; /* pointer x, y coordinates in event window */
    int x_root, y_root; /* coordinates relative to root */
}
The window member is set to the window on which the EnterNotify or LeaveNotify event was generated and is referred to as the event window. This is the window used by the X server to report the event, and is relative to the root window on which the event occurred. The root member is set to the root window of the screen on which the event occurred.

For a LeaveNotify event, if a child of the event window contains the initial position of the pointer, the subwindow component is set to that child. Otherwise, the X server sets the subwindow member to None. For an EnterNotify event, if a child of the event window contains the final pointer position, the subwindow component is set to that child or None.

The time member is set to the time when the event was generated and is expressed in milliseconds. The x and y members are set to the coordinates of the pointer position in the event window. This position is always the pointer's final position, not its initial position. If the event window is on the same screen as the root window, x and y are the pointer coordinates relative to the event window's origin. Otherwise, x and y are set to zero. The x_root and y_root members are set to the pointer's coordinates relative to the root window's origin at the time of the event.

The same_screen member is set to indicate whether the event window is on the same screen as the root window and can be either True or False. If True, the event and root windows are on the same screen. If False, the event and root windows are not on the same screen.

The focus member is set to indicate whether the event window is the focus window or an inferior of the focus window. The X server can set this member to either True or False. If True, the event window is the focus window or an inferior of the focus window. If False, the event window is not the focus window or an inferior of the focus window.

The state member is set to indicate the state of the pointer buttons and modifier keys just prior to the event. The X server can set this member to the bitwise inclusive OR of one or more of the button or modifier key masks: Button1Mask, Button2Mask, Button3Mask, Button4Mask, Button5Mask, ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask.

The mode member is set to indicate whether the events are normal events, pseudo-motion events when a grab activates, or pseudo-motion events when a grab deactivates. The X server can set this member to NotifyNormal, NotifyGrab, or NotifyUngrab.

The detail member is set to indicate the notify detail and can be NotifyAncestor, NotifyVirtual, NotifyInferior, NotifyNonlinear, or NotifyNonlinearVirtual.

10.6.1. Normal Entry/Exit Events

EnterNotify and LeaveNotify events are generated when the pointer moves from one window to another window. Normal events are identified by XEnterWindowEvent or XLeaveWindowEvent structures whose mode member is set to NotifyNormal.

- When the pointer moves from window A to window B and A is an inferior of B, the X server does the following:
- It generates a LeaveNotify event on window A, with the detail member of the \texttt{XLeaveWindowEvent} structure set to NotifyAncestor.
- It generates a LeaveNotify event on each window between window A and window B, exclusive, with the detail member of each \texttt{XLeaveWindowEvent} structure set to NotifyVirtual.
- It generates an EnterNotify event on window B, with the detail member of the \texttt{XEnterWindowEvent} structure set to NotifyInferior.
- When the pointer moves from window A to window B and B is an inferior of A, the X server does the following:
  - It generates a LeaveNotify event on window A, with the detail member of the \texttt{XLeaveWindowEvent} structure set to NotifyInferior.
  - It generates an EnterNotify event on each window between window A and window B, exclusive, with the detail member of each \texttt{XEnterWindowEvent} structure set to NotifyVirtual.
  - It generates an EnterNotify event on window B, with the detail member of the \texttt{XEnterWindowEvent} structure set to NotifyAncestor.
- When the pointer moves from window A to window B and window C is their least common ancestor, the X server does the following:
  - It generates a LeaveNotify event on window A, with the detail member of the \texttt{XLeaveWindowEvent} structure set to NotifyNonlinear.
  - It generates a LeaveNotify event on each window between window A and window C, exclusive, with the detail member of each \texttt{XLeaveWindowEvent} structure set to NotifyNonlinearVirtual.
  - It generates an EnterNotify event on each window between window C and window B, exclusive, with the detail member of each \texttt{XEnterWindowEvent} structure set to NotifyNonlinearVirtual.
  - It generates an EnterNotify event on window B, with the detail member of the \texttt{XEnterWindowEvent} structure set to NotifyNonlinear.
- When the pointer moves from window A to window B on different screens, the X server does the following:
  - It generates a LeaveNotify event on window A, with the detail member of the \texttt{XLeaveWindowEvent} structure set to NotifyNonlinear.
  - If window A is not a root window, it generates a LeaveNotify event on each window above window A up to and including its root, with the detail member of each \texttt{XLeaveWindowEvent} structure set to NotifyNonlinearVirtual.
  - If window B is not a root window, it generates an EnterNotify event on each window from window B’s root down to but not including window B, with the detail member of each \texttt{XEnterWindowEvent} structure set to NotifyNonlinearVirtual.
  - It generates an EnterNotify event on window B, with the detail member of the \texttt{XEnterWindowEvent} structure set to NotifyNonlinear.

### 10.6.2. Grab and Ungrab Entry/Exit Events

Pseudo-motion mode EnterNotify and LeaveNotify events are generated when a pointer grab activates or deactivates. Events in which the pointer grab activates are identified by \texttt{XEnterWindowEvent} or \texttt{XLeaveWindowEvent} structures whose mode member is set to NotifyGrab. Events in which the pointer grab deactivates are identified by \texttt{XEnterWindowEvent} or \texttt{XLeaveWindowEvent} structures whose mode member is set to NotifyUngrab (see \texttt{XGrabPointer}).

167
Xlib – C Library

When a pointer grab activates 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, the X server does the following:

- It generates EnterNotify and LeaveNotify events (see section 10.6.1) with the mode members of the XEnterWindowEvent and XLeaveWindowEvent structures set to NotifyGrab. These events are generated as if the pointer were to suddenly warp from its current position in $P$ to some position in $G$. However, the pointer does not warp, and the X server uses the pointer position as both the initial and final positions for the events.

When a pointer grab deactivates 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, the X server does the following:

- It generates EnterNotify and LeaveNotify events (see section 10.6.1) with the mode members of the XEnterWindowEvent and XLeaveWindowEvent structures set to NotifyUngrab. These events are generated as if the pointer were to suddenly warp from some position in $G$ to its current position in $P$. However, the pointer does not warp, and the X server uses the current pointer position as both the initial and final positions for the events.

10.7. Input Focus Events

This section describes the processing that occurs for the input focus events FocusIn and FocusOut. The X server can report FocusIn or FocusOut events to clients wanting information about when the input focus changes. The keyboard is always attached to some window (typically, the root window or a top-level window), which is called the focus window. The focus window and the position of the pointer determine the window that receives keyboard input. Clients may need to know when the input focus changes to control highlighting of areas on the screen.

To receive FocusIn or FocusOut events, set the FocusChangeMask bit in the event-mask attribute of the window.

The structure for these event types contains:

typedef struct {
    int type;          /* FocusIn or FocusOut */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event;  /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window window;    /* window of event */
    int mode;         /* NotifyNormal, NotifyGrab, NotifyUngrab */
    int detail;       /* NotifyAncestor, NotifyVirtual, NotifyInferior,
                      * NotifyNonlinear, NotifyNonlinearVirtual, NotifyPointerRoot,
                      * NotifyPointerRoot, NotifyDetailNone */
} XFocusChangeEvent;

typedef XFocusChangeEvent XFocusInEvent;
typedef XFocusChangeEvent XFocusOutEvent;

The window member is set to the window on which the FocusIn or FocusOut event was generated. This is the window used by the X server to report the event. The mode member is set to indicate whether the focus events are normal focus events, focus events while grabbed, focus events when a grab activates, or focus events when a grab deactivates. The X server can set the mode member to NotifyNormal, NotifyWhileGrabbed, NotifyGrab, or NotifyUngrab.
All FocusOut events caused by a window unmap are generated after any UnmapNotify event; however, the X protocol does not constrain the ordering of FocusOut events with respect to generated EnterNotify, LeaveNotify, VisibilityNotify, and Expose events.

Depending on the event mode, the detail member is set to indicate the notify detail and can be NotifyAncestor, NotifyVirtual, NotifyInferior, NotifyNonlinear, NotifyNonlinearVirtual, NotifyPointer, NotifyPointerRoot, or NotifyDetailNone.

10.7.1. Normal Focus Events and Focus Events While Grabbed

Normal focus events are identified by XFocuSelEvent or XFocuSelEvent structures whose mode member is set to NotifyNormal. Focus events while grabbed are identified by XFocuSelEvent or XFocuSelEvent structures whose mode member is set to NotifyWhileGrabbed. The X server processes normal focus and focus events while grabbed according to the following:

- When the focus moves from window A to window B, A is an inferior of B, and the pointer is in window P, the X server does the following:
  - It generates a FocusOut event on window A, with the detail member of the XFocuSelEvent structure set to NotifyAncestor.
  - It generates a FocusOut event on each window between window A and window B, exclusive, with the detail member of each XFocuSelEvent structure set to NotifyVirtual.
  - It generates a FocusIn event on window B, with the detail member of the XFocuSelEvent structure set to NotifyInferior.
  - If window P is an inferior of window B but window P is not window A or an inferior or ancestor of window A, it generates a FocusIn event on each window below window B, down to and including window P, with the detail member of each XFocuSelEvent structure set to NotifyPointer.

- When the focus moves from window A to window B, B is an inferior of A, and the pointer is in window P, the X server does the following:
  - If window P is an inferior of window A but P is not an inferior of window B or an ancestor of B, it generates a FocusOut event on each window from window P up to but not including window A, with the detail member of each XFocuSelEvent structure set to NotifyPointer.
  - It generates a FocusOut event on window A, with the detail member of the XFocuSelEvent structure set to NotifyInferior.
  - It generates a FocusIn event on each window between window A and window B, exclusive, with the detail member of each XFocuSelEvent structure set to NotifyVirtual.
  - It generates a FocusIn event on window B, with the detail member of the XFocuSelEvent structure set to NotifyAncestor.

- When the focus moves from window A to window B, window C is their least common ancestor, and the pointer is in window P, the X server does the following:
  - If window P is an inferior of window A, it generates a FocusOut event on each window from window P up to but not including window A, with the detail member of the XFocuSelEvent structure set to NotifyPointer.
  - It generates a FocusOut event on window A, with the detail member of the XFocuSelEvent structure set to NotifyNonlinear.
  - It generates a FocusOut event on each window between window A and window C, exclusive, with the detail member of each XFocuSelEvent structure set to NotifyNonlinearVirtual.
- It generates a FocusIn event on each window between C and B, exclusive, with the detail member of each XFocusInEvent structure set to NotifyNonlinearVirtual.
- It generates a FocusIn event on window B, with the detail member of the XFocusInEvent structure set to NotifyNonlinear.
- If window P is an inferior of window B, it generates a FocusIn event on each window below window B down to and including window P, with the detail member of the XFocusInEvent structure set to NotifyPointer.

- When the focus moves from window A to window B on different screens and the pointer is in window P, the X server does the following:
  - If window P is an inferior of window A, it generates a FocusOut event on each window from window P up to but not including window A, with the detail member of each XFocusOutEvent structure set to NotifyPointer.
  - It generates a FocusOut event on window A, with the detail member of the XFocusOutEvent structure set to NotifyNonlinear.
  - If window A is not a root window, it generates a FocusOut event on each window above window A up to and including its root, with the detail member of each XFocusOutEvent structure set to NotifyNonlinearVirtual.
  - If window B is not a root window, it generates a FocusIn event on each window below window B down to but not including window B, with the detail member of each XFocusInEvent structure set to NotifyNonlinearVirtual.
  - If window P is an inferior of window B, it generates a FocusIn event on each window below window B down to and including window P, with the detail member of each XFocusInEvent structure set to NotifyPointer.

- When the focus moves from window A to PointerRoot (events sent to the window under the pointer) or None (discard), and the pointer is in window P, the X server does the following:
  - If window P is an inferior of window A, it generates a FocusOut event on each window from window P up to but not including window A, with the detail member of each XFocusOutEvent structure set to NotifyPointer.
  - It generates a FocusOut event on window A, with the detail member of the XFocusOutEvent structure set to NotifyNonlinear.
  - If window A is not a root window, it generates a FocusOut event on each window above window A up to and including its root, with the detail member of each XFocusOutEvent structure set to NotifyNonlinearVirtual.
  - It generates a FocusIn event on the root window of all screens, with the detail member of each XFocusInEvent structure set to NotifyPointerRoot (or NotifyDetailNone).
  - If the new focus is PointerRoot, it generates a FocusIn event on each window from window P's root down to and including window P, with the detail member of each XFocusInEvent structure set to NotifyPointer.

- When the focus moves from PointerRoot (events sent to the window under the pointer) or None to window A, and the pointer is in window P, the X server does the following:
  - If the old focus is PointerRoot, it generates a FocusOut event on each window from window P up to and including window P's root, with the detail member of each XFocusOutEvent structure set to NotifyPointer.
It generates a FocusOut event on all root windows, with the detail member of each XFocusOutEvent structure set to NotifyPointerRoot (or NotifyDetailNone).

If window A is not a root window, it generates a FocusIn event on each window from window A's root down to but not including window A, with the detail member of each XFocusInEvent structure set to NotifyNonlinearVirtual.

It generates a FocusIn event on window A, with the detail member of the XFocusInEvent structure set to NotifyNonlinear.

If window P is an inferior of window A, it generates a FocusIn event on each window below window A down to and including window P, with the detail member of each XFocusInEvent structure set to NotifyPointer.

When the focus moves from PointerRoot (events sent to the window under the pointer) to None (or vice versa), and the pointer is in window P, the X server does the following:

If the old focus is PointerRoot, it generates a FocusOut event on each window from window P up to and including window P's root, with the detail member of each XFocusOutEvent structure set to NotifyPointer.

It generates a FocusOut event on all root windows, with the detail member of each XFocusOutEvent structure set to either NotifyPointerRoot or NotifyDetailNone.

It generates a FocusIn event on all root windows, with the detail member of each XFocusInEvent structure set to NotifyDetailNone or NotifyPointerRoot.

If the new focus is PointerRoot, it generates a FocusIn event on each window from window P's root down to and including window P, with the detail member of each XFocusInEvent structure set to NotifyPointer.

10.7.2. Focus Events Generated by Grabs

Focus events in which the keyboard grab activates are identified by XFocusInEvent or XFocusOutEvent structures whose mode member is set to NotifyGrab. Focus events in which the keyboard grab deactivates are identified by XFocusInEvent or XFocusOutEvent structures whose mode member is set to NotifyUngrab (see XGrabKeyboard).

When a keyboard grab activates before generating any actual KeyPress event that activates the grab, G is the grab_window, and F is the current focus, the X server does the following:

It generates FocusIn and FocusOut events, with the mode members of the XFocusInEvent and XFocusOutEvent structures set to NotifyGrab. These events are generated as if the focus were to change from F to G.

When a keyboard grab deactivates after generating any actual KeyRelease event that deactivates the grab, G is the grab_window, and F is the current focus, the X server does the following:

It generates FocusIn and FocusOut events, with the mode members of the XFocusInEvent and XFocusOutEvent structures set to NotifyUngrab. These events are generated as if the focus were to change from G to F.

10.8. Key Map State Notification Events

The X server can report KeymapNotify events to clients that want information about changes in their keyboard state.

To receive KeymapNotify events, set the KeymapStateMask bit in the event-mask attribute of the window. The X server generates this event immediately after every EnterNotify and FocusIn event.
The structure for this event type contains:
/* generated on EnterWindow and FocusIn when KeymapState selected */
typedef struct {
    int type;  /* KeymapNotify */
    unsigned long serial;  /* # of last request processed by server */
    Bool send_event;  /* true if this came from a SendEvent request */
    Display *display;  /* Display the event was read from */
    Window window;
    char key_vector[32];
} XKeymapEvent;

The window member is not used but is present to aid some toolkits. The key_vector member is set to the bit vector 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.

10.9. Exposure Events
The X protocol does not guarantee to preserve the contents of window regions when the windows are obscured or reconfigured. Some implementations may preserve the contents of windows. Other implementations are free to destroy the contents of windows when exposed. X expects client applications to assume the responsibility for restoring the contents of an exposed window region. (An exposed window region describes a formerly obscured window whose region becomes visible.) Therefore, the X server sends Expose events describing the window and the region of the window that has been exposed. A naive client application usually redraws the entire window. A more sophisticated client application redraws only the exposed region.

10.9.1. Expose Events
The X server can report Expose events to clients wanting information about when the contents of window regions have been lost. The circumstances in which the X server generates Expose events are not as definite as those for other events. However, the X server never generates Expose events on windows whose class you specified as InputOnly. The X server can generate Expose events when no valid contents are available for regions of a window and either the regions are visible, the regions are viewable and the server is (perhaps newly) maintaining backing store on the window, or the window is not viewable but the server is (perhaps newly) honoring the window’s backing-store attribute of Always or WhenMapped. The regions decompose into an (arbitrary) set of rectangles, and an Expose event is generated for each rectangle. For any given window, the X server guarantees to report contiguously all of the regions exposed by some action that causes Expose events, such as raising a window.

To receive Expose events, set the ExposureMask bit in the event-mask attribute of the window.

The structure for this event type contains:
typedef struct {
    int type;  /* Expose */
    unsigned long serial;  /* # of last request processed by server */
    Bool send_event;  /* true if this came from a SendEvent request */
    Display *display;  /* Display the event was read from */
    Window window;
    int x, y;
    int width, height;
    int count;  /* if nonzero, at least this many more */
} XExposeEvent;
The window member is set to the exposed (damaged) window. The x and y members are set to the coordinates relative to the window’s origin and indicate the upper-left corner of the rectangle. The width and height members are set to the size (extent) of the rectangle. The count member is set to the number of Expose events that are to follow. If count is zero, no more Expose events follow for this window. However, if count is nonzero, at least that number of Expose events (and possibly more) follow for this window. Simple applications that do not want to optimize redisplay by distinguishing between subareas of its window can just ignore all Expose events with nonzero counts and perform full redisplay on events with zero counts.

10.9.2. GraphicsExpose and NoExpose Events

The X server can report GraphicsExpose events to clients wanting information about when a destination region could not be computed during certain graphics requests: XCopyArea or XCopyPlane. The X server generates this event whenever a destination region could not be computed due to an obscured or out-of-bounds source region. In addition, the X server guarantees to report contiguously all of the regions exposed by some graphics request (for example, copying an area of a drawable to a destination drawable).

The X server generates a NoExpose event whenever a graphics request that might produce a GraphicsExpose event does not produce any. In other words, the client is really asking for a GraphicsExpose event but instead receives a NoExpose event.

To receive GraphicsExpose or NoExpose events, you must first set the graphics-exposure attribute of the graphics context to True. You also can set the graphics-expose attribute when creating a graphics context using XCreateGC or by calling XSetGraphicsExposures.

The structures for these event types contain:

```c
typedef struct {
    int type; /* GraphicsExpose */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Drawable drawable;
    int x, y; /* if nonzero, at least this many more */
    int width, height;
    int count; /* core is CopyArea or CopyPlane */
    int major_code;
    int minor_code; /* not defined in the core */
} XGraphicsExposeEvent;

typedef struct {
    int type; /* NoExpose */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Drawable drawable;
    int major_code; /* core is CopyArea or CopyPlane */
    int minor_code; /* not defined in the core */
} XNoExposeEvent;
```

Both structures have these common members: drawable, major_code, and minor_code. The drawable member is set to the drawable of the destination region on which the graphics request was to be performed. The major_code member is set to the graphics request initiated by the client and can be either X_CopyArea or X_CopyPlane. If it is X_CopyArea, a call to XCopyArea initiated the request. If it is X_CopyPlane, a call to XCopyPlane initiated the request. These constants are defined in <X11/Xproto.h>. The minor_code member, like the major_code member, indicates which graphics request was initiated by the client. However, the
minor code member is not defined by the core X protocol and will be zero in these cases, although it may be used by an extension.

The XGraphicsExposeEvent structure has these additional members: x, y, width, height, and count. The x and y members are set to the coordinates relative to the drawable's origin and indicate the upper-left corner of the rectangle. The width and height members are set to the size (extent) of the rectangle. The count member is set to the number of GraphicsExpose events to follow. If count is zero, no more GraphicsExpose events follow for this window. However, if count is nonzero, at least that number of GraphicsExpose events (and possibly more) are to follow for this window.

10.10. Window State Change Events

The following sections discuss:
- CirculateNotify events
- ConfigureNotify events
- CreateNotify events
- DestroyNotify events
- GravityNotify events
- MapNotify events
- MappingNotify events
- ReparentNotify events
- UnmapNotify events
- VisibilityNotify events

10.10.1. CirculateNotify Events

The X server can report CirculateNotify events to clients wanting information about when a window changes its position in the stack. The X server generates this event type whenever a window is actually restacked as a result of a client application calling XCirculateSubwindows, XCirculateSubwindowsUp, or XCirculateSubwindowsDown.

To receive CirculateNotify events, set the StructureNotifyMask bit in the event-mask attribute of the window or the SubstructureNotifyMask bit in the event-mask attribute of the parent window (in which case, circulating any child generates an event).

The structure for this event type contains:

typedef struct {
    int type; /* CirculateNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window event;
    Window window;
    int place; /* PlaceOnTop, PlaceOnBottom */
} XCirculateEvent;

The event member is set either to the restacked window or to its parent, depending on whether StructureNotify or SubstructureNotify was selected. The window member is set to the window that was restacked. The place member is set to the window's position after the restack occurs and is either PlaceOnTop or PlaceOnBottom. If it is PlaceOnTop, the window is now on top of all siblings. If it is PlaceOnBottom, the window is now below all siblings.
10.10.2. ConfigureNotify Events

The X server can report ConfigureNotify events to clients wanting information about actual changes to a window's state, such as size, position, border, and stacking order. The X server generates this event type whenever one of the following configure window requests made by a client application actually completes:

- A window's size, position, border, and/or stacking order is reconfigured by calling XConfigureWindow.
- The window's position in the stacking order is changed by calling XLowerWindow, XRaiseWindow, or XRestackWindows.
- A window is moved by calling XMoveWindow.
- A window's size is changed by calling XResizeWindow.
- A window's size and location is changed by calling XMoveResizeWindow.
- A window is mapped and its position in the stacking order is changed by calling XMapRaised.
- A window's border width is changed by calling XSetWindowBorderWidth.

To receive ConfigureNotify events, set the StructureNotifyMask bit in the event-mask attribute of the window or the SubstructureNotifyMask bit in the event-mask attribute of the parent window (in which case, configuring any child generates an event).

The structure for this event type contains:

```c
typedef struct {
    int type;    /* ConfigureNotify */
    unsigned long serial;    /* # of last request processed by server */
    Bool send_event;    /* true if this came from a SendEvent request */
    Display *display;    /* Display the event was read from */
    Window event;
    Window window;
    int x, y;
    int width, height;
    int border_width;
    Window above;
    Bool override_redirect;
} XConfigureEvent;
```

The event member is set either to the reconfigured window or to its parent, depending on whether StructureNotify or SubstructureNotify was selected. The window member is set to the window whose size, position, border, and/or stacking order was changed.

The x and y members are set to the coordinates relative to the parent window's origin and indicate the position of the upper-left outside corner of the window. The width and height members are set to the inside size of the window, not including the border. The border_width member is set to the width of the window's border, in pixels.

The above member is set to the sibling window and is used for stacking operations. If the X server sets this member to None, the window whose state was changed is on the bottom of the stack with respect to sibling windows. However, if this member is set to a sibling window, the window whose state was changed is placed on top of this sibling window.

The override_redirect member is set to the override-redirect attribute of the window. Window manager clients normally should ignore this window if the override_redirect member is True.

10.10.3. CreateNotify Events

The X server can report CreateNotify events to clients wanting information about creation of windows. The X server generates this event whenever a client application creates a window.
by calling \texttt{XCreateWindow} or \texttt{XCreateSimpleWindow}.

To receive \texttt{CreateNotify} events, set the \texttt{SubstructureNotifyMask} bit in the event-mask attribute of the window. Creating any children then generates an event.

The structure for the event type contains:

\begin{verbatim}
  typedef struct {
    int type;           /* CreateNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event;    /* true if this came from a SendEvent request */
    Display *display;   /* Display the event was read from */
    Window parent;      /* parent of the window */
    Window window;      /* window id of window created */
    int x, y;           /* window location */
    int width, height;  /* size of window */
    int border_width;   /* border width */
    Bool override_redirect; /* creation should be overridden */
  } XCreateWindowEvent;
\end{verbatim}

The parent member is set to the created window's parent. The window member specifies the created window. The x and y members are set to the created window's coordinates relative to the parent window's origin and indicate the position of the upper-left outside corner of the created window. The width and height members are set to the inside size of the created window (not including the border) and are always nonzero. The border_width member is set to the width of the created window's border, in pixels. The override_redirect member is set to the override-redirect attribute of the window. Window manager clients normally should ignore this window if the override_redirect member is True.

10.10.4. DestroyNotify Events

The X server can report \texttt{DestroyNotify} events to clients wanting information about which windows are destroyed. The X server generates this event whenever a client application destroys a window by calling \texttt{XDestroyWindow} or \texttt{XDestroySubwindows}.

The ordering of the \texttt{DestroyNotify} events is such that for any given window, \texttt{DestroyNotify} is generated on all inferiors of the window before being generated on the window itself. The X protocol does not constrain the ordering among siblings and across subhierarchies.

To receive \texttt{DestroyNotify} events, set the \texttt{StructureNotifyMask} bit in the event-mask attribute of the window or the \texttt{SubstructureNotifyMask} bit in the event-mask attribute of the parent window (in which case, destroying any child generates an event).

The structure for this event type contains:

\begin{verbatim}
  typedef struct {
    int type;           /* DestroyNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event;    /* true if this came from a SendEvent request */
    Display *display;   /* Display the event was read from */
    Window event;       /* window that is destroyed */
    Window window;      /* window that is destroyed */
  } XDestroyWindowEvent;
\end{verbatim}

The event member is set either to the destroyed window or to its parent, depending on whether \texttt{StructureNotify} or \texttt{SubstructureNotify} was selected. The window member is set to the window that is destroyed.
10.10.5. GravityNotify Events

The X server can report GravityNotify events to clients wanting information about when a window is moved because of a change in the size of its parent. The X server generates this event whenever a client application actually moves a child window as a result of resizing its parent by calling XConfigureWindow, XMoveResizeWindow, or XResizeWindow.

To receive GravityNotify events, set the StructureNotifyMask bit in the event-mask attribute of the window or the SubstructureNotifyMask bit in the event-mask attribute of the parent window (in which case, any child that is moved because its parent has been resized generates an event).

The structure for this event type contains:

```c
typedef struct {
    int type; /* GravityNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window event; /* Window the event occurred in */
    Window window; /* Window the event occurred in */
    int x, y; /* position of the upper-left outside corner of the window */
} XGravityEvent;
```

The event member is set either to the window that was moved or to its parent, depending on whether StructureNotify or SubstructureNotify was selected. The window member is set to the child window that was moved. The x and y members are set to the coordinates relative to the new parent window’s origin and indicate the position of the upper-left outside corner of the window.

10.10.6. MapNotify Events

The X server can report MapNotify events to clients wanting information about which windows are mapped. The X server generates this event type whenever a client application changes the window’s state from unmapped to mapped by calling XMapWindow, XMapRaised, XMapSubwindows, XReparentWindow, or as a result of save-set processing.

To receive MapNotify events, set the StructureNotifyMask bit in the event-mask attribute of the window or the SubstructureNotifyMask bit in the event-mask attribute of the parent window (in which case, mapping any child generates an event).

The structure for this event type contains:

```c
typedef struct {
    int type; /* MapNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window event; /* Window the event occurred in */
    Window window; /* Window the event occurred in */
    Bool override_redirect; /* boolean, is override set... */
} XMapEvent;
```

The event member is set either to the window that was mapped or to its parent, depending on whether StructureNotify or SubstructureNotify was selected. The window member is set to the window that was mapped. The override_redirect member is set to the override-redirect attribute of the window. Window manager clients normally should ignore this window if the override-redirect attribute is True, because these events usually are generated from pop-ups, which override structure control.
10.10.7. MappingNotify Events

The X server reports MappingNotify events to all clients. There is no mechanism to express disinterest in this event. The X server generates this event type whenever a client application successfully calls:

- `XSetModifierMapping` to indicate which KeyCodes are to be used as modifiers
- `XChangeKeyboardMapping` to change the keyboard mapping
- `XSetPointerMapping` to set the pointer mapping

The structure for this event type contains:

```c
typedef struct {
    int type; /* MappingNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window window; /* one of MappingModifier, MappingKeyboard, MappingPointer */
    int request; /* first keycode */
    int first_keycode; /* defines range of change w. first_keycode*/
    int count;
} XMappingEvent;
```

The request member is set to indicate the kind of mapping change that occurred and can be `MappingModifier`, `MappingKeyboard`, `MappingPointer`. If it is `MappingModifier`, the modifier mapping was changed. If it is `MappingKeyboard`, the keyboard mapping was changed. If it is `MappingPointer`, the pointer button mapping was changed. The `first_keycode` and `count` members are set only if the request member was set to `MappingKeyboard`. The number in `first_keycode` represents the first number in the range of the altered mapping, and `count` represents the number of keycodes altered.

To update the client application's knowledge of the keyboard, you should call `XRefreshKeyboardMapping`.

10.10.8. ReparentNotify Events

The X server can report ReparentNotify events to clients wanting information about changing a window’s parent. The X server generates this event whenever a client application calls `XReparentWindow` and the window is actually reparented.

To receive ReparentNotify events, set the `StructureNotifyMask` bit in the event-mask attribute of the window or the `SubstructureNotifyMask` bit in the event-mask attribute of either the old or the new parent window (in which case, reparenting any child generates an event).

The structure for this event type contains:

```c
typedef struct {
    int type; /* ReparentNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window event;
    Window window;
    Window parent;
    int x, y;
    Bool override_redirect;
} XReparentEvent;
```

The event member is set either to the reparented window or to the old or the new parent, depending on whether `StructureNotify` or `SubstructureNotify` was selected. The window
member is set to the window that was reparented. The parent member is set to the new parent window. The x and y members are set to the reparented window’s coordinates relative to the new parent window’s origin and define the upper-left outer corner of the reparented window. The override_redirect member is set to the override-redirect attribute of the window specified by the window member. Window manager clients normally should ignore this window if the override_redirect member is True.

10.10.9. UnmapNotify Events

The X server can report UnmapNotify events to clients wanting information about which windows are unmapped. The X server generates this event type whenever a client application changes the window’s state from mapped to unmapped.

To receive UnmapNotify events, set the StructureNotifyMask bit in the event-mask attribute of the window or the SubstructureNotifyMask bit in the event-mask attribute of the parent window (in which case, unmapping any child window generates an event).

The structure for this event type contains:

```c
typedef struct {
    int type; /* UnmapNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window event;
    Window window;
    Bool from_configure;
} XUnmapEvent;
```

The event member is set either to the unmapped window or to its parent, depending on whether StructureNotify or SubstructureNotify was selected. This is the window used by the X server to report the event. The window member is set to the window that was unmapped. The from_configure member is set to True if the event was generated as a result of a resizing of the window’s parent when the window itself had a win_gravity of UnmapGravity.

10.10.10. VisibilityNotify Events

The X server can report VisibilityNotify events to clients wanting any change in the visibility of the specified window. A region of a window is visible if someone looking at the screen can actually see it. The X server generates this event whenever the visibility changes state. However, this event is never generated for windows whose class is InputOnly.

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

To receive VisibilityNotify events, set the VisibilityChangeMask bit in the event-mask attribute of the window.

The structure for this event type contains:

```c
typedef struct {
    int type; /* VisibilityNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
} XVisibilityEvent;
```
The window member is set to the window whose visibility state changes. The state member is set to the state of the window's visibility and can be `VisibilityUnobscured`, `VisibilityPartiallyObscured`, or `VisibilityFullyObscured`. The X server ignores all of a window's subwindows when determining the visibility state of the window and processes VisibilityNotify events according to the following:

- When the window changes state from partially obscured, fully obscured, or not viewable to viewable and completely unobscured, the X server generates the event with the state member of the XVisibilityEvent structure set to `VisibilityUnobscured`.
- When the window changes state from viewable and completely unobscured or not viewable to viewable and partially obscured, the X server generates the event with the state member of the XVisibilityEvent structure set to `VisibilityPartiallyObscured`.
- When the window changes state from viewable and completely unobscured, viewable and partially obscured, or not viewable to viewable and fully obscured, the X server generates the event with the state member of the XVisibilityEvent structure set to `VisibilityFullyObscured`.

10.11. Structure Control Events

This section discusses:

- CirculateRequest events
- ConfigureRequest events
- MapRequest events
- ResizeRequest events

10.11.1. CirculateRequest Events

The X server can report CirculateRequest events to clients wanting information about when another client initiates a circulate window request on a specified window. The X server generates this event type whenever a client initiates a circulate window request on a window and a subwindow actually needs to be restacked. The client initiates a circulate window request on the window by calling `XCirculateSubwindows`, `XCirculateSubwindowsUp`, or `XCirculateSubwindowsDown`.

To receive CirculateRequest events, set the SubstructureRedirectMask in the event-mask attribute of the window. Then, in the future, the circulate window request for the specified window is not executed, and thus, any subwindow's position in the stack is not changed. For example, suppose a client application calls `XCirculateSubwindowsUp` to raise a subwindow to the top of the stack. If you had selected SubstructureRedirectMask on the window, the X server reports to you a CirculateRequest event and does not raise the subwindow to the top of the stack.

The structure for this event type contains:

```c
typedef struct {
    int type;  /* CirculateRequest */
    unsigned long serial;  /* # of last request processed by server */
    Bool send_event;  /* true if this came from a SendEvent request */
    Display *display;  /* Display the event was read from */
    Window parent;
    Window window;
    int place;  /* PlaceOnTop, PlaceOnBottom */
} XCirculateRequestEvent;
```
The parent member is set to the parent window. The window member is set to the subwindow to be restacked. The place member is set to what the new position in the stacking order should be and is either PlaceOnTop or PlaceOnBottom. If it is PlaceOnTop, the subwindow should be on top of all siblings. If it is PlaceOnBottom, the subwindow should be below all siblings.

10.11.2. ConfigureRequest Events

The X server can report ConfigureRequest events to clients wanting information about when a different client initiates a configure window request on any child of a specified window. The configure window request attempts to reconfigure a window's size, position, border, and stacking order. The X server generates this event whenever a different client initiates a configure window request on a window by calling XConfigureWindow, XLowerWindow, XRaiseWindow, XMapRaised, XMoveResizeWindow, XMoveWindow, XResizeWindow, XRestackWindows, or XSetWindowBorderWidth.

To receive ConfigureRequest events, set the SubstructureRedirectMask bit in the event-mask attribute of the window. ConfigureRequest events are generated when a ConfigureWindow protocol request is issued on a child window by another client. For example, suppose a client application calls XLowerWindow to lower a window. If you had selected SubstructureRedirectMask on the parent window and if the override-redirect attribute of the window is set to False, the X server reports a ConfigureRequest event to you and does not lower the specified window.

The structure for this event type contains:

typedef struct {
    int type; /* ConfigureRequest */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window parent;
    Window window;
    int x, y;
    int width, height;
    int border_width;
    Window above;
    int detail; /* Above, Below, TopIf, BottomIf, Opposite */
    unsigned long value_mask;
} XConfigureRequestEvent;

The parent member is set to the parent window. The window member is set to the window whose size, position, border width, and/or stacking order is to be reconfigured. The value_mask member indicates which components were specified in the ConfigureWindow protocol request. 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 above (sibling) and detail (stack-mode), which are reported as Above and None, respectively, if they are not given in the request.

10.11.3. MapRequest Events

The X server can report MapRequest events to clients wanting information about a different client's desire to map windows. A window is considered mapped when a map window request completes. The X server generates this event whenever a different client initiates a map window request on an unmapped window whose override_redirect member is set to False. Clients initiate map window requests by calling XMapWindow, XMapRaised, or XMapSubwindows.
To receive MapRequest events, set the SubstructureRedirectMask bit in the event-mask attribute of the window. This means another client's attempts to map a child window by calling one of the map window request functions is intercepted, and you are sent a MapRequest instead. For example, suppose a client application calls XMapWindow to map a window. If you (usually a window manager) had selected SubstructureRedirectMask on the parent window and if the override-redirect attribute of the window is set to False, the X server reports a MapRequest event to you and does not map the specified window. Thus, this event gives your window manager client the ability to control the placement of subwindows.

The structure for this event type contains:

```
typedef struct {
    int type;            /* MapRequest */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event;     /* true if this came from a SendEvent request */
    Display *display;    /* Display the event was read from */
    Window parent;
    Window window;
} XMapRequestEvent;
```

The parent member is set to the parent window. The window member is set to the window to be mapped.

10.11.4. ResizeRequest Events

The X server can report ResizeRequest events to clients wanting information about another client's attempts to change the size of a window. The X server generates this event whenever some other client attempts to change the size of the specified window by calling XConfigureWindow, XResizeWindow, or XMoveResizeWindow.

To receive ResizeRequest events, set the ResizeRedirect bit in the event-mask attribute of the window. Any attempts to change the size by other clients are then redirected.

The structure for this event type contains:

```
typedef struct {
    int type;            /* ResizeRequest */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event;     /* true if this came from a SendEvent request */
    Display *display;    /* Display the event was read from */
    Window window;
    int width, height;
} XResizeRequestEvent;
```

The window member is set to the window whose size another client attempted to change. The width and height members are set to the inside size of the window, excluding the border.

10.12. Colormap State Change Events

The X server can report ColormapNotify events to clients wanting information about when the colormap changes and when a colormap is installed or uninstalled. The X server generates this event type whenever a client application:

- Changes the colormap member of the XSetWindowAttributes structure by calling XChangeWindowAttributes, XFreeColormap, or XSetWindowColormap
- Installs or uninstalls the colormap by calling XInstallColormap or XUninstallColormap

To receive ColormapNotify events, set the ColormapChangeMask bit in the event-mask attribute of the window.
The structure for this event type contains:

```c
typedef struct {
    int type; /* ClientMessage */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window window;
    Atom message_type; /* Atom that indicates how the data should be interpreted by the receiving client. */
    int format; /* Specifies whether the data should be viewed as a list of bytes, shorts, or longs. */
    union {
        char b[20]; /* Represents data of 20 8-bit values. */
        short s[10];
        long l[5];
    } data;
} XClientMessageEvent;
```

The message_type member is set to an atom that indicates how the data should be interpreted by the receiving client. The format member is set to 8, 16, or 32 and specifies whether the data should be viewed as a list of bytes, shorts, or longs. The data member is a union that contains the members b, s, and l. The b, s, and l members represent data of 20 8-bit values,
10 16-bit values, and 5 32-bit values. Particular message types might not make use of all these values. The X server places no interpretation on the values in the window, message_type, or data members.

10.13.2. PropertyNotify Events
The X server can report PropertyNotify events to clients wanting information about property changes for a specified window.

To receive PropertyNotify events, set the PropertyChangeMask bit in the event-mask attribute of the window.

The structure for this event type contains:

typedef struct {
  int type; /* PropertyNotify */
  unsigned long serial; /* # of last request processed by server */
  Bool send_event; /* true if this came from a SendEvent request */
  Display *display; /* Display the event was read from */
  Window window;
  Atom atom;
  Time time;
  int state; /* PropertyNewValue or PropertyDelete */
} XPropertyEvent;

The window member is set to the window whose associated property was changed. The atom member is set to the property's atom and indicates which property was changed or desired. The time member is set to the server time when the property was changed. The state member is set to indicate whether the property was changed to a new value or deleted and can be PropertyNewValue or PropertyDelete. The state member is set to PropertyNewValue when a property of the window is changed using XChangeProperty or XRotateWindowProperties (even when adding zero-length data using XChangeProperty) and when replacing all or part of a property with identical data using XChangeProperty or XRotateWindowProperties. The state member is set to PropertyDelete when a property of the window is deleted using XDeleteProperty or, if the delete argument is True, XGetWindowProperty.

10.13.3. SelectionClear Events
The X server reports SelectionClear events to the client losing ownership of a selection. The X server generates this event type when another client asserts ownership of the selection by calling XSetSelectionOwner.

The structure for this event type contains:

typedef struct {
  int type; /* SelectionClear */
  unsigned long serial; /* # of last request processed by server */
  Bool send_event; /* true if this came from a SendEvent request */
  Display *display; /* Display the event was read from */
  Window window;
  Atom selection;
  Time time;
} XSelectionClearEvent;

The selection member is set to the selection atom. The time member is set to the last change time recorded for the selection. The window member is the window that was specified by the current owner (the owner losing the selection) in its XSetSelectionOwner call.
10.13.4. SelectionRequest Events

The X server reports SelectionRequest events to the owner of a selection. The X server generates this event whenever a client requests a selection conversion by calling XConvertSelection for the owned selection.

The structure for this event type contains:

```c
typedef struct {
    int type; /* SelectionRequest */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window owner;
    Window requestor;
    Atom selection;
    Atom target;
    Atom property;
    Time time;
} XSelectionRequestEvent;
```

The owner member is set to the window that was specified by the current owner in its XSetSelectionOwner call. The requestor member is set to the window requesting the selection. The selection member is set to the atom that names the selection. For example, PRIMARY is used to indicate the primary selection. The target member is set to the atom that indicates the type the selection is desired in. The property member can be a property name or None. The time member is set to the timestamp or CurrentTime value from the ConvertSelection request.

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

10.13.5. SelectionNotify Events

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

If None is specified as the property in the ConvertSelection protocol request, the owner should choose a property name, store the result as that property on the requestor window, and then send a SelectionNotify giving that actual property name.

The structure for this event type contains:

```c
typedef struct {
    int type; /* SelectionNotify */
    unsigned long serial; /* # of last request processed by server */
    Bool send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Window requestor;
    Atom selection;
    Atom target;
    Atom property; /* atom or None */
    Time time;
} XSelectionEvent;
```
The requestor member is set to the window associated with the requestor of the selection. The selection member is set to the atom that indicates the selection. For example, PRIMARY is used for the primary selection. The target member is set to the atom that indicates the converted type. For example, PIXMAP is used for a pixmap. The property member is set to the atom that indicates which property the result was stored on. If the conversion failed, the property member is set to None. The time member is set to the time the conversion took place and can be a timestamp or CurrentTime.
Chapter 11

Event Handling Functions

This chapter discusses the Xlib functions you can use to:

- Select events
- Handle the output buffer and the event queue
- Select events from the event queue
- Send and get events
- Handle protocol errors

Note

Some toolkits use their own event-handling functions and do not allow you to interchange these event-handling functions with those in Xlib. For further information, see the documentation supplied with the toolkit.

Most applications simply are event loops: they wait for an event, decide what to do with it, execute some amount of code that results in changes to the display, and then wait for the next event.

11.1. Selecting Events

There are two ways to select the events you want reported to your client application. One way is to set the event_mask member of the XSetWindowAttributes structure when you call XCreateWindow and XChangeWindowAttributes. Another way is to use XSelectInput.

XSelectInput(display, w, event_mask)

- display: Specifies the connection to the X server.
- w: Specifies the window whose events you are interested in.
- event_mask: Specifies the event mask.

The XSelectInput function requests that the X server report the events associated with the specified event mask. Initially, X will not report any of these events. Events are reported relative to a window. If a window is not interested in a device event, it usually propagates to the closest ancestor that is interested, unless the do_not_propagate mask prohibits it.

Setting the event-mask attribute of a window overrides any previous call for the same window but not for other clients. Multiple clients can select for the same events on the same window with the following restrictions:

- Multiple clients can select events on the same window because their event masks are disjoint. When the X server generates an event, it reports it to all interested clients.
- Only one client at a time can select CirculateRequest, ConfigureRequest, or MapRequest events, which are associated with the event mask SubstructureRedirectMask.
- Only one client at a time can select a ResizeRequest event, which is associated with the event mask ResizeRedirectMask.
- Only one client at a time can select a ButtonPress event, which is associated with the event mask ButtonPressMask.
The server reports the event to all interested clients. XSelectInput can generate a BadWindow error.

11.2. Handling the Output Buffer
The output buffer is an area used by Xlib to store requests. The functions described in this section flush the output buffer if the function would block or not return an event. That is, all requests residing in the output buffer that have not yet been sent are transmitted to the X server. These functions differ in the additional tasks they might perform.

To flush the output buffer, use XFlush.

\[
\text{XFlush}(\text{display})
\]

\[
\text{display} \quad \text{Specifies the connection to the X server.}
\]

The XFlush function flushes the output buffer. Most client applications need not use this function because the output buffer is automatically flushed as needed by calls to XPending, XNextEvent, and XWindowEvent. Events generated by the server may be enqueued into the library’s event queue.

To flush the output buffer and then wait until all requests have been processed, use XSync.

\[
\text{XSync}(\text{display, discard})
\]

\[
\text{display} \quad \text{Specifies the connection to the X server.}
\]
\[
\text{discard} \quad \text{Specifies a Boolean value that indicates whether XSync discards all events on the event queue.}
\]

The XSync function flushes the output buffer and then waits until all requests have been received and processed by the X server. Any errors generated must be handled by the error handler. For each protocol error received by Xlib, XSync calls the client application’s error handling routine (see section 11.8.2). Any events generated by the server are enqueued into the library’s event queue.

Finally, if you passed False, XSync does not discard the events in the queue. If you passed True, XSync discards all events in the queue, including those events that were on the queue before XSync was called. Client applications seldom need to call XSync.

11.3. Event Queue Management
Xlib maintains an event queue. However, the operating system also may be buffering data in its network connection that is not yet read into the event queue.

To check the number of events in the event queue, use XEventsQueued.

\[
\text{int XEventsQueued}(\text{display, mode})
\]

\[
\text{display} \quad \text{Specifies the connection to the X server.}
\]
\[
\text{mode} \quad \text{Specifies the mode. You can pass QueuedAlready, QueuedAfterFlush, or QueuedAfterReading.}
\]

If mode is QueuedAlready, XEventsQueued returns the number of events already in the event queue (and never performs a system call). If mode is QueuedAfterFlush, XEventsQueued returns the number of events already in the queue if the number is nonzero.
If there are no events in the queue, XEventsQueued flushes the output buffer, attempts to read more events out of the application’s connection, and returns the number read. If mode is QueuedAfterReading, XEventsQueued returns the number of events already in the queue if the number is nonzero. If there are no events in the queue, XEventsQueued attempts to read more events out of the application’s connection without flushing the output buffer and returns the number read.

XEventsQueued always returns immediately without I/O if there are events already in the queue. XEventsQueued with mode QueuedAfterFlush is identical in behavior to XPending. XEventsQueued with mode QueuedAlready is identical to the XLength function.

To return the number of events that are pending, use XPending.

```c
int XPending(Display *display)
{
    Display *display;
    Speciftes the connection to the X server.
}
```

The XPending function returns the number of events that have been received from the X server but have not been removed from the event queue. XPending is identical to XEventsQueued with the mode QueuedAfterFlush specified.

11.4. Manipulating the Event Queue

Xlib provides functions that let you manipulate the event queue. This section discusses how to:

- Obtain events, in order, and remove them from the queue
- Peek at events in the queue without removing them
- Obtain events that match the event mask or the arbitrary predicate procedures that you provide

11.4.1. Returning the Next Event

To get the next event and remove it from the queue, use XNextEvent.

```c
XNextEvent(Display *display, XEvent *event_return)
{
    Display *display;
    XEvent *event_return;
    Speciftes the connection to the X server.
    Returns the next event in the queue.
}
```

The XNextEvent function copies the first event from the event queue into the specified XEvent structure and then removes it from the queue. If the event queue is empty, XNextEvent flushes the output buffer and blocks until an event is received.

To peek at the event queue, use XPeekEvent.

```c
XPeekEvent(Display *display, event_return)
{
    Display *display;
    XEvent *event_return;
    Speciftes the connection to the X server.
    Returns a copy of the matched event’s associated structure.
}
```

The XPeekEvent function returns the first event from the event queue, but it does not remove the event from the queue. If the queue is empty, XPeekEvent flushes the output buffer and blocks until an event is received. It then copies the event into the client-supplied XEvent structure without removing it from the event queue.
11.4.2. Selecting Events Using a Predicate Procedure

Each of the functions discussed in this section requires you to pass a predicate procedure that determines if an event matches what you want. Your predicate procedure must decide only if the event is useful and must not call Xlib functions. In particular, a predicate is called from inside the event routine, which must lock data structures so that the event queue is consistent in a multi-threaded environment.

The predicate procedure and its associated arguments are:

\[
\text{Bool (} \ast \text{predicate}(\text{display, event, arg})
\]

- \text{display} \quad \text{Specifies the connection to the X server.}
- \text{event} \quad \text{Specifies the XEvent structure.}
- \text{arg} \quad \text{Specifies the argument passed in from the XIfEvent, XCheckIfEvent, or XPeekIfEvent function.}

The predicate procedure is called once for each event in the queue until it finds a match. After finding a match, the predicate procedure must return True. If it did not find a match, it must return False.

To check the event queue for a matching event and, if found, remove the event from the queue, use XIfEvent.

\[
\text{XIfEvent(} \text{display, event_return, predicate, arg)}
\]

- \text{display} \quad \text{Specifies the connection to the X server.}
- \text{event_return} \quad \text{Returns the matched event’s associated structure.}
- \text{predicate} \quad \text{Specifies the procedure that is to be called to determine if the next event in the queue matches what you want.}
- \text{arg} \quad \text{Specifies the user-supplied argument that will be passed to the predicate procedure.}

The XIfEvent function completes only when the specified predicate procedure returns True for an event, which indicates an event in the queue matches. XIfEvent flushes the output buffer if it blocks waiting for additional events. XIfEvent removes the matching event from the queue and copies the structure into the client-supplied XEvent structure.

To check the event queue for a matching event without blocking, use XCheckIfEvent.

\[
\text{Bool XCheckIfEvent(} \text{display, event_return, predicate, arg)}
\]

- \text{display} \quad \text{Specifies the connection to the X server.}
- \text{event_return} \quad \text{Returns a copy of the matched event’s associated structure.}
- \text{predicate} \quad \text{Specifies the procedure that is to be called to determine if the next event in the queue matches what you want.}
arg Specifies the user-supplied argument that will be passed to the predicate procedure.

When the predicate procedure finds a match, \texttt{XCheckIfEvent} copies the matched event into the client-supplied \texttt{XEvent} structure and returns \texttt{True}. (This event is removed from the queue.) If the predicate procedure finds no match, \texttt{XCheckIfEvent} returns \texttt{False}, and the output buffer will have been flushed. All earlier events stored in the queue are not discarded.

To check the event queue for a matching event without removing the event from the queue, use \texttt{XPeekIfEvent}.

\texttt{XPeekIfEvent(display, event\_return, predicate, arg)}

\texttt{Display *display;}
\texttt{XEvent *event\_return;}
\texttt{Bool (*predicate)();}
\texttt{XPointer arg;}

\texttt{display} Specifies the connection to the X server.
\texttt{event\_return} Returns a copy of the matched event's associated structure.
\texttt{predicate} Specifies the procedure that is to be called to determine if the next event in the queue matches what you want.
\texttt{arg} Specifies the user-supplied argument that will be passed to the predicate procedure.

The \texttt{XPeekIfEvent} function returns only when the specified predicate procedure returns \texttt{True} for an event. After the predicate procedure finds a match, \texttt{XPeekIfEvent} copies the matched event into the client-supplied \texttt{XEvent} structure without removing the event from the queue. \texttt{XPeekIfEvent} flushes the output buffer if it blocks waiting for additional events.

11.4.3. Selecting Events Using a Window or Event Mask
The functions discussed in this section let you select events by window or event types, allowing you to process events out of order.

To remove the next event that matches both a window and an event mask, use \texttt{XWindowEvent}.

\texttt{XWindowEvent(display, w, event\_mask, event\_return)}

\texttt{Display *display;}
\texttt{Window w;}
\texttt{long event\_mask;}
\texttt{XEvent *event\_return;}

\texttt{display} Specifies the connection to the X server.
\texttt{w} Specifies the window whose events you are interested in.
\texttt{event\_mask} Specifies the event mask.
\texttt{event\_return} Returns the matched event's associated structure.

The \texttt{XWindowEvent} function searches the event queue for an event that matches both the specified window and event mask. When it finds a match, \texttt{XWindowEvent} removes that event from the queue and copies it into the specified \texttt{XEvent} structure. The other events stored in the queue are not discarded. If a matching event is not in the queue, \texttt{XWindowEvent} flushes the output buffer if it blocks waiting for additional events.

To remove the next event that matches both a window and an event mask (if any), use \texttt{XCheckWindowEvent}. This function is similar to \texttt{XWindowEvent} except that it never blocks and it returns a \texttt{Bool} indicating if the event was returned.
Bool XCheckWindowEvent(display, w, event_mask, event_return)
    Display *display;
    Window w;
    long event_mask;
    XEvent *event_return;

    display    Specifies the connection to the X server.
    w          Specifies the window whose events you are interested in.
    event_mask Specifies the event mask.
    event_return Returns the matched event’s associated structure.

    The XCheckWindowEvent function searches the event queue and then the events available on
    the server connection for the first event that matches the specified window and event mask. If
    it finds a match, XCheckWindowEvent removes that event, copies it into the specified
    XEvent structure, and returns True. The other events stored in the queue are not discarded.
    If the event you requested is not available, XCheckWindowEvent returns False, and the output
    buffer will have been flushed.

    To remove the next event that matches an event mask, use XMaskEvent.

    XMaskEvent(display, event_mask, event_return)
        Display *display;
        long event_mask;
        XEvent *event_return;

        display    Specifies the connection to the X server.
        event_mask Specifies the event mask.
        event_return Returns the matched event’s associated structure.

    The XMaskEvent function searches the event queue for the events associated with the
    specified mask. When it finds a match, XMaskEvent removes that event and copies it into
    the specified XEvent structure. The other events stored in the queue are not discarded. If the
    event you requested is not in the queue, XMaskEvent flushes the output buffer and blocks
    until one is received.

    To return and remove the next event that matches an event mask (if any), use XCheck-
    MaskEvent. This function is similar to XMaskEvent except that it never blocks and it
    returns a Bool indicating if the event was returned.

    Bool XCheckMaskEvent(display, event_mask, event_return)
        Display *display;
        long event_mask;
        XEvent *event_return;

        display    Specifies the connection to the X server.
        event_mask Specifies the event mask.
        event_return Returns the matched event’s associated structure.

    The XCheckMaskEvent function searches the event queue and then any events available on
    the server connection for the first event that matches the specified mask. If it finds a match,
    XCheckMaskEvent removes that event, copies it into the specified XEvent structure, and
    returns True. The other events stored in the queue are not discarded. If the event you
    requested is not available, XCheckMaskEvent returns False, and the output buffer will have
    been flushed.

    To return and remove the next event in the queue that matches an event type, use
XCheckTypedEvent.

Bool XCheckTypedEvent(display, event_type, event_return)
    Display *display;
    int event_type;
    XEvent *event_return;

display Specifies the connection to the X server.
event_type Specifies the event type to be compared.
event_return Returns the matched event's associated structure.

The XCheckTypedEvent function searches the event queue and then any events available on
the server connection for the first event that matches the specified type. If it finds a match,
XCheckTypedEvent removes that event, copies it into the specified XEvent structure, and
returns True. The other events in the queue are not discarded. If the event is not available,
XCheckTypedEvent returns False, and the output buffer will have been flushed.

To return and remove the next event in the queue that matches an event type and a window,
use XCheckTypedWindowEvent.

Bool XCheckTypedWindowEvent(display, w, event_type, event_return)
    Display *display;
    Window w;
    int event_type;
    XEvent *event_return;

display Specifies the connection to the X server.
w Specifies the window.
event_type Specifies the event type to be compared.
event_return Returns the matched event's associated structure.

The XCheckTypedWindowEvent function searches the event queue and then any events
available on the server connection for the first event that matches the specified type and win¬
dow. If it finds a match, XCheckTypedWindowEvent removes the event from the queue,
copies it into the specified XEvent structure, and returns True. The other events in the queue
are not discarded. If the event is not available, XCheckTypedWindowEvent returns False,
and the output buffer will have been flushed.

11.5. Putting an Event Back into the Queue

To push an event back into the event queue, use XPutBackEvent.

XPutBackEvent(display, event)
    Display *display;
    XEvent *event;

display Specifies the connection to the X server.
event Specifies the event.

The XPutBackEvent function pushes an event back onto the head of the display's event
queue by copying the event into the queue. This can be useful if you read an event and then
decide that you would rather deal with it later. There is no limit to the number of times in
succession that you can call XPutBackEvent.
11.6. Sending Events to Other Applications

To send an event to a specified window, use XSendEvent. This function is often used in selection processing. For example, the owner of a selection should use XSendEvent to send a SelectionNotify event to a requestor when a selection has been converted and stored as a property.

Status XSendEvent(display, w, propagate, event_mask, event_send)

- Display *display;
- Window w;
- Bool propagate;
- long event_mask;
- XEvent *event_send;

display Specifies the connection to the X server.
w Specifies the window the event is to be sent to, PointerWindow, or Input-Focus.
propagate Specifies a Boolean value.
event_mask Specifies the event mask.
event_send Specifies the event that is to be sent.

The XSendEvent function identifies the destination window, determines which clients should receive the specified events, and ignores any active grabs. This function requires you to pass an event mask. For a discussion of the valid event mask names, see section 10.3. This function uses the w argument to identify the destination window as follows:

- If w is PointerWindow, the destination window is the window that contains the pointer.
- If w is InputFocus and if the focus window contains the pointer, the destination window is the window that contains the pointer; otherwise, the destination window is the focus window.

To determine which clients should receive the specified events, XSendEvent uses the propagate argument as follows:

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

The event in the XEvent structure must be one of the core events or one of the events defined by an extension (or a BadValue error results) so that the X server can correctly byte-swap the contents as necessary. The contents of the event are otherwise unaltered and unchecked by the X server except to force send_event to True in the forwarded event and to set the serial number in the event correctly.

XSendEvent returns zero if the conversion to wire protocol format failed and returns nonzero otherwise.

XSendEvent can generate BadValue and BadWindow errors.
11.7. Getting Pointer Motion History

Some X server implementations will maintain a more complete history of pointer motion than is reported by event notification. The pointer position at each pointer hardware interrupt may be stored in a buffer for later retrieval. This buffer is called the motion history buffer. For example, a few applications, such as paint programs, want to have a precise history of where the pointer traveled. However, this historical information is highly excessive for most applications.

To determine the approximate maximum number of elements in the motion buffer, use XDisplayMotionBufferSize.

\[
\text{unsigned long } \text{XDisplayMotionBufferSize}(\text{display})
\]

\[\text{Display * display;}\]

\[\text{display} \quad \text{Specifies the connection to the X server.}\]

The server may retain the recent history of the pointer motion and do so to a finer granularity than is reported by MotionNotify events. The XGetMotionEvents function makes this history available.

To get the motion history for a specified window and time, use XGetMotionEvents.

\[
\text{XTimeCoord * XGetMotionEvents}(\text{display}, \text{w}, \text{start}, \text{stop}, \text{nevents_return})
\]

\[\text{Display * display;}\]

\[\text{Window w;}\]

\[\text{Time start, stop;}\]

\[\text{int *nevents_return;}\]

\[\text{display} \quad \text{Specifies the connection to the X server.}\]

\[\text{w} \quad \text{Specifies the window.}\]

\[\text{start} \quad \text{stop} \quad \text{Specify the time interval in which the events are returned from the motion history buffer. You can pass a timestamp or CurrentTime.}\]

\[\text{nevents_return} \quad \text{Returns the number of events from the motion history buffer.}\]

The XGetMotionEvents function returns all events in the motion history buffer that fall between the specified start and stop times, inclusive, and that have coordinates that lie within the specified window (including its borders) at its present placement. If the server does not support motion history, or if the start time is later than the stop time, or if the start time is in the future, no events are returned, and XGetMotionEvents returns NULL. If the stop time is in the future, it is equivalent to specifying CurrentTime. The return type for this function is a structure defined as follows:

\[
\text{typedef struct}{
    \text{Time time;
    short x, y;
} \text{XTimeCoord;}\]

The time member is set to the time, in milliseconds. The x and y members are set to the coordinates of the pointer and are reported relative to the origin of the specified window. To free the data returned from this call, use XFree.

XGetMotionEvents can generate a BadWindow error.

11.8. Handling Protocol Errors

Xlib provides functions that you can use to enable or disable synchronization and to use the default error handlers.
11.8.1. Enabling or Disabling Synchronization
When debugging X applications, it often is very convenient to require Xlib to behave synchro-
nously so that errors are reported as they occur. The following function lets you disable or
enable synchronous behavior. Note that graphics may occur 30 or more times more slowly
when synchronization is enabled. On POSIX-conformant systems, there is also a global vari-
able _Xdebug that, if set to nonzero before starting a program under a debugger, will force
synchronous library behavior.

After completing their work, all Xlib functions that generate protocol requests call what is
known as an after function. XSetAfterFunction sets which function is to be called.

```c
int (*XSetAfterFunction(display, procedure))();
```

`display` Specifies the connection to the X server.

`procedure` Specifies the function to be called.

The specified procedure is called with only a display pointer. XSetAfterFunction returns the
previous after function.

To enable or disable synchronization, use XSynchronize.

```c
int (*XSynchronize(display, onoff))();
```

`display` Specifies the connection to the X server.

`onoff` Specifies a Boolean value that indicates whether to enable or disable synchro-
nization.

The XSynchronize function returns the previous after function. If onoff is True, XSyn-
chronize turns on synchronous behavior. If onoff is False, XSynchronize turns off synchro-
nous behavior.

11.8.2. Using the Default Error Handlers
There are two default error handlers in Xlib: one to handle typically fatal conditions (for exam-
ple, the connection to a display server dying because a machine crashed) and one to handle
protocol errors from the X server. These error handlers can be changed to user-supplied rou-
tines if you prefer your own error handling and can be changed as often as you like. If either
function is passed a NULL pointer, it will invoke the default handler. The action of the
default handlers is to print an explanatory message and exit.

To set the error handler, use XSetErrorHandler.

```c
int (*XSetErrorHandler(handler))();
```

`handler` Specifies the program’s supplied error handler.

Xlib generally calls the program’s supplied error handler whenever an error is received. It is
not called on BadName errors from OpenFont, LookupColor, or AllocNamedColor proto-
col requests or on BadFont errors from a QueryFont protocol request. These errors gen-
erally are reflected back to the program through the procedural interface. Because this condi-
tion is not assumed to be fatal, it is acceptable for your error handler to return. However, the
error handler should not call any functions (directly or indirectly) on the display that will gen-
erate protocol requests or that will look for input events. The previous error handler is
returned.

The XErrorEvent structure contains:
typedef struct {
    int type;
    Display *display;  /* Display the event was read from */
    unsigned long serial;  /* serial number of failed request */
    unsigned char error_code;  /* error code of failed request */
    unsigned char request_code;  /* Major op-code of failed request */
    unsigned char minor_code;  /* Minor op-code of failed request */
    XID resourceid;  /* resource id */
} XErrorEvent;

The serial member is the number of requests, starting from one, sent over the network connection since it was opened. It is the number that was the value of NextRequest immediately before the failing call was made. The request_code member is a protocol request of the procedure that failed, as defined in `<X11/Xproto.h>`. The following error codes can be returned by the functions described in this chapter:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadAccess</td>
<td>A client attempts to grab a key/button combination already grabbed by another client. A client attempts to free a colormap entry that it had not already allocated, or to free an entry in a colormap that was created with all entries writable. A client attempts to store into a read-only or unallocated colormap entry. A client attempts to modify the access control list from other than the local (or otherwise authorized) host. A client attempts to select an event type that another client has already selected.</td>
</tr>
<tr>
<td>BadAlloc</td>
<td>The server fails to allocate the requested resource. Note that the explicit listing of BadAlloc errors in requests only covers allocation errors at a very coarse level and is not intended to (nor can it in practice hope to) cover all cases of a server running out of allocation space in the middle of service. The semantics when a server runs out of allocation space are left unspecified, but a server may generate a BadAlloc error on any request for this reason, and clients should be prepared to receive such errors and handle or discard them.</td>
</tr>
<tr>
<td>BadAtom</td>
<td>A value for an atom argument does not name a defined atom.</td>
</tr>
<tr>
<td>BadColor</td>
<td>A value for a colormap argument does not name a defined colormap.</td>
</tr>
<tr>
<td>BadCursor</td>
<td>A value for a cursor argument does not name a defined cursor.</td>
</tr>
<tr>
<td>BadDrawable</td>
<td>A value for a drawable argument does not name a defined window or pixmap.</td>
</tr>
<tr>
<td>BadFont</td>
<td>A value for a font argument does not name a defined font (or, in some cases, GContext).</td>
</tr>
<tr>
<td>BadGC</td>
<td>A value for a GContext argument does not name a defined GContext.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BadIDChoice</td>
<td>The value chosen for a resource identifier either is not included in the range assigned to the client or is already in use. Under normal circumstances, this cannot occur and should be considered a server or Xlib error.</td>
</tr>
<tr>
<td>BadImplementation</td>
<td>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.</td>
</tr>
<tr>
<td>BadLength</td>
<td>The length of a request is shorter or longer than that required to contain the arguments. This is an internal Xlib or server error. The length of a request exceeds the maximum length accepted by the server.</td>
</tr>
<tr>
<td>BadMatch</td>
<td>In a graphics request, the root and depth of the graphics context does not match that of the drawable. An InputOnly window is used as a drawable. Some argument or pair of arguments has the correct type and range, but it fails to match in some other way required by the request. An InputOnly window lacks this attribute.</td>
</tr>
<tr>
<td>BadName</td>
<td>A font or color of the specified name does not exist.</td>
</tr>
<tr>
<td>BadPixmap</td>
<td>A value for a pixmap argument does not name a defined pixmap.</td>
</tr>
<tr>
<td>BadRequest</td>
<td>The major or minor opcode does not specify a valid request. This usually is an Xlib or server error.</td>
</tr>
<tr>
<td>BadValue</td>
<td>Some numeric value falls outside of 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).</td>
</tr>
<tr>
<td>BadWindow</td>
<td>A value for a window argument does not name a defined window.</td>
</tr>
</tbody>
</table>

**Note**

The BadAtom, BadColor, BadCursor, BadDrawable, BadFont, BadGC, BadPixmap, and BadWindow errors are also used when the argument type is extended by a set of fixed alternatives.

To obtain textual descriptions of the specified error code, use `XGetErrorText`.

```c
XGetErrorText(display, code, buffer_return, length)

Display *display;
int code;
char *buffer_return;
int length;
```
display    Specifies the connection to the X server.
code       Specifies the error code for which you want to obtain a description.
buffer_return Returns the error description.
length     Specifies the size of the buffer.

The XGetErrorText function copies a null-terminated string describing the specified error code into the specified buffer. The returned text is in the encoding of the current locale. It is recommended that you use this function to obtain an error description because extensions to Xlib may define their own error codes and error strings.

To obtain error messages from the error database, use XGetErrorDatabaseText.

XGetErrorDatabaseText(display, name, message, default_string, buffer_return, length)

Display *display;
char *name, *message;
char *default_string;
char *buffer_return;
int length;

display    Specifies the connection to the X server.
name       Specifies the name of the application.
message    Specifies the type of the error message.
default_string Specifies the default error message if none is found in the database.
buffer_return Returns the error description.
length     Specifies the size of the buffer.

The XGetErrorDatabaseText function returns a null-terminated message (or the default message) from the error message database. Xlib uses this function internally to look up its error messages. The default_string is assumed to be in the encoding of the current locale. The buffer_return text is in the encoding of the current locale.

The name argument should generally be the name of your application. The message argument should indicate which type of error message you want. If the name and message are not in the Host Portable Character Encoding the result is implementation dependent. Xlib uses three predefined "application names" to report errors (uppercase and lowercase matter):

XProtoError The protocol error number is used as a string for the message argument.
XlibMessage These are the message strings that are used internally by the library.
XRequest For a core protocol request, the major request protocol number is used for the message argument. For an extension request, the extension name (as given by InitExtension) followed by a period (.) and the minor request protocol number is used for the message argument. If no string is found in the error database, the default_string is returned to the buffer argument.

To report an error to the user when the requested display does not exist, use XDisplayName.

char *XDisplayName(string)
char *string;

string    Specifies the character string.

The XDisplayName function returns the name of the display that XOpenDisplay would attempt to use. If a NULL string is specified, XDisplayName looks in the environment for the display and returns the display name that XOpenDisplay would attempt to use. This makes it easier to report to the user precisely which display the program attempted to open when the initial connection attempt failed.
To handle fatal I/O errors, use `XSetIOErrorHandler`.

```c
int (*XSetIOErrorHandler(handler))()
    int (*handler)(Display *);
```

`handler` Specifies the program's supplied error handler.

The `XSetIOErrorHandler` sets the fatal I/O error handler. Xlib calls the program's supplied error handler if any sort of system call error occurs (for example, the connection to the server was lost). This is assumed to be a fatal condition, and the called routine should not return. If the I/O error handler does return, the client process exits.

Note that the previous error handler is returned.
Chapter 12

Input Device Functions

You can use the Xlib input device functions to:

- Grab the pointer and individual buttons on the pointer
- Grab the keyboard and individual keys on the keyboard
- Move the pointer
- Set the input focus
- Manipulate the keyboard and pointer settings
- Manipulate the keyboard encoding

12.1. Pointer Grabbing

Xlib provides functions that you can use to control input from the pointer, which usually is a mouse. Usually, as soon as keyboard and mouse events occur, the X server delivers them to the appropriate client, which is determined by the window and input focus. The X server provides sufficient control over event delivery to allow window managers to support mouse ahead and various other styles of user interface. Many of these user interfaces depend upon synchronous delivery of events. The delivery of pointer and keyboard events can be controlled independently.

When mouse buttons or keyboard keys are grabbed, events will be sent to the grabbing client rather than the normal client who would have received the event. If the keyboard or pointer is in asynchronous mode, further mouse and keyboard events will continue to be processed. If the keyboard or pointer is in synchronous mode, no further events are processed until the grabbing client allows them (see XAllowEvents). The keyboard or pointer is considered frozen during this interval. The event that triggered the grab can also be replayed.

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.

There are two kinds of grabs: active and passive. An active grab occurs when a single client grabs the keyboard and/or pointer explicitly (see XGrabPointer and XGrabKeyboard). A passive grab occurs when clients grab a particular keyboard key or pointer button in a window, and the grab will activate when the key or button is actually pressed. Passive grabs are convenient for implementing reliable pop-up menus. For example, you can guarantee that the pop-up is mapped before the up pointer button event occurs by grabbing a button requesting synchronous behavior. The down event will trigger the grab and freeze further processing of pointer events until you have the chance to map the pop-up window. You can then allow further event processing. The up event will then be correctly processed relative to the pop-up window.

For many operations, there are functions that take a time argument. The X server includes a timestamp in various events. One special time, called CurrentTime, represents the current server time. The X server maintains the time when the input focus was last changed, when the keyboard was last grabbed, when the pointer was last grabbed, or when a selection was last changed. Your application may be slow reacting to an event. You often need some way to specify that your request should not occur if another application has in the meanwhile taken control of the keyboard, pointer, or selection. By providing the timestamp from the event in the request, you can arrange that the operation not take effect if someone else has performed an operation in the meanwhile.
A timestamp is a time value, expressed in milliseconds. It typically is the time since the last server reset. Timestamp values wrap around (after about 49.7 days). The server, given its current time is represented by timestamp T, always interprets timestamps from clients by treating half of the timestamp space as being later in time than T. One timestamp value, named \texttt{CurrentTime}, is never generated by the server. This value is reserved for use in requests to represent the current server time.

For many functions in this section, you pass pointer event mask bits. The valid pointer event mask bits are: \texttt{ButtonPressMask}, \texttt{ButtonReleaseMask}, \texttt{EnterWindowMask}, \texttt{LeaveWindowMask}, \texttt{PointerMotionMask}, \texttt{PointerMotionHintMask}, \texttt{Button1MotionMask}, \texttt{Button2MotionMask}, \texttt{Button3MotionMask}, \texttt{Button4MotionMask}, \texttt{Button5MotionMask}, \texttt{ButtonMotionMask}, and \texttt{KeyMapStateMask}. For other functions in this section, you pass keymask bits. The valid keymask bits are: \texttt{ShiftMask}, \texttt{LockMask}, \texttt{ControlMask}, \texttt{Mod1Mask}, \texttt{Mod2Mask}, \texttt{Mod3Mask}, \texttt{Mod4Mask}, and \texttt{Mod5Mask}.

To grab the pointer, use \texttt{XGrabPointer}.

```c
int XGrabPointer(display, grab_window, owner_events, event_mask, pointer_mode,
    keyboard_mode, confine_to, cursor, time)
```

- \texttt{display} Specifies the connection to the X server.
- \texttt{grab_window} Specifies the grab window.
- \texttt{owner_events} Specifies a Boolean value that indicates whether the pointer events are to be reported as usual or reported with respect to the grab window if selected by the event mask.
- \texttt{event_mask} Specifies which pointer events are reported to the client. The mask is the bitwise inclusive OR of the valid pointer event mask bits.
- \texttt{pointer_mode} Specifies further processing of pointer events. You can pass \texttt{GrabModeSync} or \texttt{GrabModeAsync}.
- \texttt{keyboard_mode} Specifies further processing of keyboard events. You can pass \texttt{GrabModeSync} or \texttt{GrabModeAsync}.
- \texttt{confine_to} Specifies the window to confine the pointer in or \texttt{None}.
- \texttt{cursor} Specifies the cursor that is to be displayed during the grab or \texttt{None}.
- \texttt{time} Specifies the time. You can pass either a timestamp or \texttt{CurrentTime}.

The \texttt{XGrabPointer} function actively grabs control of the pointer and returns \texttt{GrabSuccess} if the grab was successful. Further pointer events are reported only to the grabbing client. \texttt{XGrabPointer} overrides any active pointer grab by this client. If \texttt{owner_events} is \texttt{False}, all generated pointer events are reported with respect to \texttt{grab_window} and are reported only if selected by \texttt{event_mask}. If \texttt{owner_events} is \texttt{True} and if a generated pointer event would normally be reported to this client, it is reported as usual. Otherwise, the event is reported with respect to the \texttt{grab_window} and is reported only if selected by \texttt{event_mask}. For either value of \texttt{owner_events}, unreported events are discarded.

If the \texttt{pointer_mode} is \texttt{GrabModeAsync}, pointer event processing continues as usual. If the pointer is currently frozen by this client, the processing of events for the pointer is resumed. If the \texttt{pointer_mode} is \texttt{GrabModeSync}, the state of the pointer, as seen by client applications,
appears to freeze, and the X server generates no further pointer events until the grabbing client calls XAllowEvents or until the pointer grab is released. Actual pointer changes are not lost while the pointer is frozen; they are simply queued in the server for later processing.

If the keyboard_mode is GrabModeAsync, keyboard event processing is unaffected by activation of the grab. If the keyboard_mode is GrabModeSync, the state of the keyboard, as seen by client applications, appears to freeze, and the X server generates no further keyboard events until the grabbing client calls XAllowEvents or until the pointer grab is released. Actual keyboard changes are not lost while the pointer is frozen; they are simply queued in the server for later processing.

If a cursor is specified, it is displayed regardless of what window the pointer is in. If None is specified, the normal cursor for that window is displayed when the pointer is in grab_window or one of its subwindows; otherwise, the cursor for grab_window is displayed.

If a confine_to window is specified, the pointer is restricted to stay contained in that window. The confine_to window need have no relationship to the grab_window. If the pointer is not initially in the confine_to window, it is warped automatically to the closest edge just before the grab activates and enter/leave events are generated as usual. If the confine_to window is subsequently reconfigured, the pointer is warped automatically, as necessary, to keep it contained in the window.

The time argument allows you to avoid certain circumstances that come up if applications take a long time to respond or if there are long network delays. Consider a situation where you have two applications, both of which normally grab the pointer when clicked on. If both applications specify the timestamp from the event, the second application may wake up faster and successfully grab the pointer before the first application. The first application then will get an indication that the other application grabbed the pointer before its request was processed.

XGrabPointer generates EnterNotify and LeaveNotify events.

Either 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, XGrabPointer fails and returns GrabNotViewable. If the pointer is actively grabbed by some other client, it fails and returns AlreadyGrabbed. If the pointer is frozen by an active grab of another client, it fails and returns GrabFrozen. If the specified time is earlier than the last-pointer-grab time or later than the current X server time, it fails and returns GrabInvalidTime. Otherwise, the last-pointer-grab time is set to the specified time (CurrentTime is replaced by the current X server time).

XGrabPointer can generate BadCursor, BadValue, and BadWindow errors.

To ungrab the pointer, use XUngrabPointer.

XUngrabPointer(display, time)
   Display *display;
   Time time;

display Specifies the connection to the X server.
time Specifies the time. You can pass either a timestamp or CurrentTime.

The XUngrabPointer function releases the pointer and any queued events if this client has actively grabbed the pointer from XGrabPointer, XGrabButton, or from a normal button press. XUngrabPointer does not release the pointer if the specified time is earlier than the last-pointer-grab time or is later than the current X server time. It also generates EnterNotify and LeaveNotify events. The X server performs an UnGrabPointer request 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.
To change an active pointer grab, use `XChangeActivePointerGrab`.

```
XChangeActivePointerGrab(display, event_mask, cursor, time)
```

- `display`: Specifies the connection to the X server.
- `event_mask`: Specifies which pointer events are reported to the client. The mask is the bitwise inclusive OR of the valid pointer event mask bits.
- `cursor`: Specifies the cursor that is to be displayed or `None`.
- `time`: Specifies the time. You can pass either a timestamp or `CurrentTime`.

The `XChangeActivePointerGrab` function changes the specified dynamic parameters if the pointer is actively grabbed by the client and if the specified time is no earlier than the last-pointer-grab time and no later than the current X server time. This function has no effect on the passive parameters of a `XGrabButton`. The interpretation of `event_mask` and `cursor` is the same as described in `XGrabPointer`.

`XChangeActivePointerGrab` can generate `BadCursor` and `BadValue` errors.

To grab a pointer button, use `XGrabButton`.

```
XGrabButton(display, button, modifiers, grab_window, owner_events, event_mask,
            pointer_mode, keyboard_mode, confine_to, cursor)
```

- `display`: Specifies the connection to the X server.
- `button`: Specifies the pointer button that is to be grabbed or `AnyButton`.
- `modifiers`: Specifies the set of keymasks or `AnyModifier`. The mask is the bitwise inclusive OR of the valid keymask bits.
- `grab_window`: Specifies the grab window.
- `owner_events`: Specifies a Boolean value that indicates whether the pointer events are to be reported as usual or reported with respect to the grab window if selected by the event mask.
- `event_mask`: Specifies which pointer events are reported to the client. The mask is the bitwise inclusive OR of the valid pointer event mask bits.
- `pointer_mode`: Specifies further processing of pointer events. You can pass `GrabModeSync` or `GrabModeAsync`.
- `keyboard_mode`: Specifies further processing of keyboard events. You can pass `GrabModeSync` or `GrabModeAsync`.
- `confine_to`: Specifies the window to confine the pointer in or `None`.
- `cursor`: Specifies the cursor that is to be displayed or `None`.

The `XGrabButton` function establishes a passive grab. In the future, the pointer is actively grabbed (as for `XGrabPointer`), the last-pointer-grab time is set to the time at which the
button was pressed (as transmitted in the ButtonPress event), and the ButtonPress 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 as for XGrabPointer. The active grab is terminated automatically when the logical state of the pointer has all buttons released (independent of the state of the logical modifier keys).

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.

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

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

XGrabButton can generate BadCursor, BadValue, and BadWindow errors.

To ungrab a pointer button, use XUngrabButton.

XUngrabButton(display, button, modifiers, grab_window)
- Display *display; Specifies the connection to the X server.
- unsigned int button; Specifies the pointer button that is to be released or AnyButton.
- unsigned int modifiers; Specifies the set of keymasks or AnyModifier. The mask is the bitwise inclusive OR of the valid keymask bits.
- Window grab_window; Specifies the grab window.

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

XUngrabButton can generate BadValue and BadWindow errors.

12.2. Keyboard Grabbing

Xlib provides functions that you can use to grab or ungrab the keyboard as well as allow events.

For many functions in this section, you pass keymask bits. The valid keymask bits are: ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, and
ModSMask.

To grab the keyboard, use XGrabKeyboard.

```c
def XGrabKeyboard(display, grab_window, owner_events, pointer_mode, keyboard_mode, time)
    Display *display;
    Window grab_window;
    Bool owner_events;
    int pointer_mode, keyboard_mode;
    Time time;

display    Specifies the connection to the X server.
grab_window Specifies the grab window.
owner_events Specifies a Boolean value that indicates whether the keyboard events are to be reported as usual.
pointer_mode Specifies further processing of pointer events. You can pass GrabModeSync or GrabModeAsync.
keyboard_mode Specifies further processing of keyboard events. You can pass GrabModeSync or GrabModeAsync.
time    Specifies the time. You can pass either a timestamp or CurrentTime.
```

The XGrabKeyboard function actively grabs control of the keyboard and generates FocusIn and FocusOut events. Further key events are reported only to the grabbing client. XGrabKeyboard 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 the keyboard_mode argument is GrabModeAsync, keyboard event processing continues as usual. If the keyboard is currently frozen by this client, then processing of keyboard events is resumed. If the keyboard_mode argument is GrabModeSync, the state of the keyboard (as seen by client applications) appears to freeze, and the X server generates no further keyboard events until the grabbing client issues a releasing XAllowEvents call or until the keyboard grab is released. Actual keyboard changes are not lost while the keyboard is frozen; they are simply queued in the server for later processing.

If pointer_mode is GrabModeAsync, pointer event processing is unaffected by activation of the grab. If pointer_mode is GrabModeSync, the state of the pointer (as seen by client applications) appears to freeze, and the X server generates no further pointer events until the grabbing client issues a releasing XAllowEvents call or until the keyboard grab is released. Actual pointer changes are not lost while the pointer is frozen; they are simply queued in the server for later processing.

If the keyboard is actively grabbed by some other client, XGrabKeyboard fails and returns AlreadyGrabbed. If grab_window is not viewable, it fails and returns GrabNotViewable. If the keyboard is frozen by an active grab of another client, it fails and returns GrabFrozen. If the specified time is earlier than the last-keyboard-grab time or later than the current X server time, it fails and returns GrabInvalidTime. Otherwise, the last-keyboard-grab time is set to the specified time (CurrentTime is replaced by the current X server time).

XGrabKeyboard can generate BadValue and BadWindow errors.

To ungrab the keyboard, use XUngrabKeyboard.
XUngrabKeyboard(display, time)
  Display *display;
  Time time;

display specifies the connection to the X server.

time specifies the time. You can pass either a timestamp or CurrentTime.

The XUngrabKeyboard function releases the keyboard and any queued events if this client has it actively grabbed from either XGrabKeyboard or XGrabKey. XUngrabKeyboard does not release the keyboard and any queued events if the specified time is earlier than the last-keyboard-grab time or is later than the current X server time. It also generates FocusIn and FocusOut events. The X server automatically performs an UngrabKeyboard request if the event window for an active keyboard grab becomes not viewable.

To passively grab a single key of the keyboard, use XGrabKey.

XGrabKey(display, keycode, modifiers, grab_window, owner_events, pointer_mode, keyboard_mode)
  Display *display;
  int keycode;
  unsigned int modifiers;
  Window grab_window;
  Bool owner_events;
  int pointer_mode, keyboard_mode;

display specifies the connection to the X server.

keycode specifies the KeyCode or AnyKey.

modifiers specifies the set of keymasks or AnyModifier. The mask is the bitwise inclusive OR of the valid keymask bits.

grab_window specifies the grab window.

owner_events specifies a Boolean value that indicates whether the keyboard events are to be reported as usual.

pointer_mode specifies further processing of pointer events. You can pass GrabModeSync or GrabModeAsync.

keyboard_mode specifies further processing of keyboard events. You can pass GrabModeSync or GrabModeAsync.

The XGrabKey function establishes a passive grab on the keyboard. In the future, the keyboard is actively grabbed (as for XGrabKeyboard), the last-keyboard-grab time is set to the time at which the key was pressed (as transmitted in the KeyPress event), and the KeyPress event is reported 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 descendant 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 as for XGrabKeyboard. The active grab is terminated automatically when the logical state of the keyboard has the specified key released (independent of the logical state of the modifier keys).

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.
A modifiers argument 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 keycode argument of AnyKey is equivalent to issuing the request for all possible KeyCodes. Otherwise, the specified keycode must be in the range specified by min_keycode and max_keycode in the connection setup, or a BadValue error results.

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

XGrabKey can generate BadAccess, BadValue, and BadWindow errors.

To ungrab a key, use XUngrabKey.

XUngrabKey(display, keycode, modifiers, grab-window)

Display *display;
int keycode;
unsigned int modifiers;
Window grab-window;

display Specifies the connection to the X server.
keycode Specifies the KeyCode or AnyKey.
modifiers Specifies the set of keymasks or AnyModifier. The mask is the bitwise inclusive OR of the valid keymask bits.

grab-window Specifies the grab window.

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

XUngrabKey can generate BadValue and BadWindow errors.

12.3. Resuming Event Processing

The previous sections discussed grab mechanisms with which processing of events by the server can be temporarily suspended. This section describes the mechanism for resuming event processing.

To allow further events to be processed when the device has been frozen, use XAllowEvents.

XAllowEvents(display, event_mode, time)

Display *display;
int event_mode;
Time time;

display Specifies the connection to the X server.
event_mode Specifies the event mode. You can pass AsyncPointer, SyncPointer, AsyncKeyboard, SyncKeyboard, ReplayPointer, ReplayKeyboard, AsyncBoth, or SyncBoth.
time Specifies the time. You can pass either a timestamp or CurrentTime.

The XAllowEvents function releases some queued events if the client has caused a device to freeze. It has no effect if the specified time is earlier than the last-grab time of the most recent active grab for the client or if the specified time is later than the current X server time. Depending on the event_mode argument, the following occurs:
AsyncPointer
If the pointer is frozen by the client, pointer event processing continues as usual. If the pointer is frozen twice by the client on behalf of two separate grabs, AsyncPointer thaws for both. AsyncPointer has no effect if the pointer is not frozen by the client, but the pointer need not be grabbed by the client.

SyncPointer
If the pointer is frozen and actively grabbed by the client, pointer event processing continues as usual until the next ButtonPress or ButtonRelease event is reported to the client. At this time, the pointer again appears to freeze. However, if the reported event causes the pointer grab to be released, the pointer does not freeze. SyncPointer has no effect if the pointer is not frozen by the client or if the pointer is not grabbed by the client.

ReplayPointer
If the pointer is actively grabbed by the client and is frozen as the result of an event having been sent to the client (either from the activation of a XGrabButton or from a previous XAllowEvents with mode SyncPointer but not from a XGrabPointer), the pointer grab is released and that event is completely reprocessed. This time, however, the function ignores any passive grabs at or above (towards the root of) the grab_window of the grab just released. The request has no effect if the pointer is not grabbed by the client or if the pointer is not frozen as the result of an event.

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

SyncKeyboard
If the keyboard is frozen and actively grabbed by the client, keyboard event processing continues as usual until the next KeyPress or KeyRelease event is reported to the client. At this time, the keyboard again appears to freeze. However, if the reported event causes the keyboard grab to be released, the keyboard does not freeze. SyncKeyboard has no effect if the keyboard is not frozen by the client or if the keyboard is not grabbed by the client.

ReplayKeyboard
If the keyboard is actively grabbed by the client and is frozen as the result of an event having been sent to the client (either from the activation of a XGrabKey or from a previous XAllowEvents with mode SyncKeyboard but not from a XGrabKeyboard), the keyboard grab is released and that event is completely reprocessed. This time, however, the function ignores any passive grabs at or above (towards the root of) the grab_window of the grab just released. The request has no effect if the keyboard is not grabbed by the client or if the keyboard is not frozen as the result of an event.
SyncBoth

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

AsyncBoth

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

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

XAllowEvents can generate a BadValue error.

12.4. Moving the Pointer

Although movement of the pointer normally should be left to the control of the end user, sometimes it is necessary to move the pointer to a new position under program control.

To move the pointer to an arbitrary point in a window, use XWarpPointer.

XWarpPointer(display, src_w, dest_w, src_x, src_y, src_width, src_height, dest_x, dest_y)

- display Specifies the connection to the X server.
- src_w Specifies the source window or None.
- dest_w Specifies the destination window or None.
- src_x, src_y Specify a rectangle in the source window.
- src_width, src_height
- dest_x, dest_y Specify the x and y coordinates within the destination window.

If dest_w is None, XWarpPointer moves the pointer by the offsets (dest_x, dest_y) relative to the current position of the pointer. If dest_w is a window, XWarpPointer moves the
pointer to the offsets (dest_x, dest_y) relative to the origin of dest_w. However, if src_w is a window, the move only takes place if the window src_w contains the pointer and if the specified rectangle of src_w contains the pointer.

The src_x and src_y coordinates are relative to the origin of src_w. If src_height is zero, it is replaced with the current height of src_w minus src_y. If src_width is zero, it is replaced with the current width of src_w minus src_x.

There is seldom any reason for calling this function. The pointer should normally be left to the user. If you do use this function, however, it generates events just as if the user had instantaneously moved the pointer from one position to another. Note that you cannot use XWarpPointer to move the pointer outside the confine_to window of an active pointer grab. An attempt to do so will only move the pointer as far as the closest edge of the confine_to window.

XWarpPointer can generate a BadWindow error.

12.5. Controlling Input Focus

Xlib provides functions that you can use to set and get the input focus. The input focus is a shared resource, and cooperation among clients is required for correct interaction. See the Inter-Client Communication Conventions Manual for input focus policy.

To set the input focus, use XSetInputFocus.

XSetInputFocus(display, focus, revert_to, time)
Display *display;
Window focus;
int revert_to;
Time time;

display Specifies the connection to the X server.
focus Specifies the window, PointerRoot, or None.
revert_to Specifies where the input focus reverts to if the window becomes not viewable. You can pass RevertToParent, RevertToPointerRoot, or RevertToNone.
time Specifies the time. You can pass either a timestamp or CurrentTime.

The XSetInputFocus function changes the input focus and the last-focus-change time. It has no effect if the specified time is earlier than the current last-focus-change time or is later than the current X server time. Otherwise, the last-focus-change time is set to the specified time (CurrentTime is replaced by the current X server time). XSetInputFocus causes the X server to generate FocusIn and FocusOut events.

Depending on the focus argument, the following occurs:

- If focus is None, all keyboard events are discarded until a new focus window is set, and the revert_to argument is ignored.
- If focus is a window, 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 as usual. Otherwise, the event is reported relative to the focus window.
- If focus is PointerRoot, the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event. In this case, the revert_to argument is ignored.

The specified focus window must be viewable at the time XSetInputFocus is called, or a BadMatch error results. If the focus window later becomes not viewable, the X server evaluates the revert_to argument to determine the new focus window as follows:

- If revert_to is RevertToParent, the focus reverts to the parent (or the closest viewable ancestor), and the new revert_to value is taken to be RevertToNone.
If revert_to is RevertToPointerRoot or RevertToNone, the focus reverts to PointerRoot or None, respectively. When the focus reverts, the X server generates FocusIn and FocusOut events, but the last-focus-change time is not affected.

XSetInputFocus can generate BadMatch, BadValue, and BadWindow errors.

To obtain the current input focus, use XGetInputFocus.

XGetInputFocus(display, focus_return, revert_to_return)
  Display *display;
  Window *focus_return;
  int *revert_to_return;

display Specifies the connection to the X server.
focus_return Returns the focus window, PointerRoot, or None.
revert_to_return Returns the current focus state (RevertToParent, RevertToPointerRoot, or RevertToNone).

The XGetInputFocus function returns the focus window and the current focus state.

## 12.6. Keyboard and Pointer Settings

Xlib provides functions that you can use to change the keyboard control, obtain a list of the auto-repeat keys, turn keyboard auto-repeat on or off, ring the bell, set or obtain the pointer button or keyboard mapping, and obtain a bit vector for the keyboard.

This section discusses the user-preference options of bell, key click, pointer behavior, and so on. The default values for many of these functions are determined by command line arguments to the X server and, on POSIX-conformant systems, are typically set in the /etc/ttys file. Not all implementations will actually be able to control all of these parameters.

The XChangeKeyboardControl function changes control of a keyboard and operates on a XKeyboardControl structure:

```c
/* Mask bits for ChangeKeyboardControl */
#define KBKeyClickPercent (1L<<0)
#define KBBellPercent (1L<<1)
#define KBBellPitch (1L<<2)
#define KBBellDuration (1L<<3)
#define KBLed (1L<<4)
#define KBLedMode (1L<<5)
#define KBKey (1L<<6)
#define KBAutoRepeatMode (1L<<7)
*/

typedef struct {
  int key_click_percent;
  int bell_percent;
  int bell_pitch;
  int bell_duration;
  int led;
  int led_mode; /* LedModeOn, LedModeOff */
  int key;
  int auto_repeat_mode; /* AutoRepeatModeOff, AutoRepeatModeOn, AutoRepeatModeDefault */
} XKeyboardControl;
```
The key_click_percent member sets the volume for key clicks between 0 (off) and 100 (loud) inclusive, if possible. A setting of -1 restores the default. Other negative values generate a BadValue error.

The bell_percent sets the base volume for the bell between 0 (off) and 100 (loud) inclusive, if possible. A setting of -1 restores the default. Other negative values generate a BadValue error. The bell_pitch member sets the pitch (specified in Hz) of the bell, if possible. A setting of -1 restores the default. Other negative values generate a BadValue error. The bell_duration member sets the duration of the bell specified in milliseconds, if possible. A setting of -1 restores the default. Other negative values generate a BadValue error.

If both the led_mode and led members are specified, the state of that LED is changed, if possible. The led_mode member can be set to LedModeOn or LedModeOff. If only led_mode is specified, the state of all LEDs are changed, if possible. At most 32 LEDs numbered from one are supported. No standard interpretation of LEDs is defined. If led is specified without led_mode, a BadMatch error results.

If both the auto_repeat_mode and key members are specified, the auto_repeat_mode of that key is changed (according to AutoRepeatModeOn, AutoRepeatModeOff, or AutoRepeatModeDefault), if possible. If only auto_repeat_mode is specified, the global auto_repeat_mode for the entire keyboard is changed, if possible, and does not affect the per key settings. If a key is specified without an auto_repeat_mode, a BadMatch error results. Each key has an individual mode of whether or not it should auto-repeat and a default setting for the 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 global mode is AutoRepeatModeOn, keys should obey their individual auto-repeat modes. When global mode is AutoRepeatModeOff, 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 its auto-repeat setting.

A bell generator connected with the console but not directly on a 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.

XChangeKeyboardControl(display, value_mask, values)
Display *display;
unsigned long value_mask;
XKeyboardControl *values;
display Speaks the connection to the X server.
value_mask Speaks which controls to change. This mask is the bitwise inclusive OR of the valid control mask bits.
values Speaks one value for each bit set to 1 in the mask.

The XChangeKeyboardControl function controls the keyboard characteristics defined by the XKeyboardControl structure. The value_mask argument specifies which values are to be changed.

XChangeKeyboardControl can generate BadMatch and BadValue errors.

To obtain the current control values for the keyboard, use XGetKeyboardControl.

XGetKeyboardControl(display, values_return)
Display *display;
XKeyboardState *values_return;
display Speaks the connection to the X server.
values_return  Returns the current keyboard controls in the specified XKeyboardState structure.

The XGetKeyboardControl function returns the current control values for the keyboard to the XKeyboardState structure.

typedef struct {
    int key_click_percent;
    int bell_percent;
    unsigned int bell_pitch, bell_duration;
    unsigned long led_mask;
    int global_auto_repeat;
    char auto_repeats[32];
} XKeyboardState;

For the LEDs, the least-significant bit of led_mask corresponds to LED one, and each bit set to 1 in led_mask indicates an LED that is lit. The global_auto_repeat member can be set to AutoRepeatModeOn or AutoRepeatModeOff. The auto_repeats member is a bit vector. Each bit set to 1 indicates that auto-repeat is enabled for the corresponding key. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7 with the least-significant bit in the byte representing key 8N.

To turn on keyboard auto-repeat, use XAutoRepeatOn.

XAutoRepeatOn(display)
    Display *display;

display  Specifies the connection to the X server.

The XAutoRepeatOn function turns on auto-repeat for the keyboard on the specified display.

To turn off keyboard auto-repeat, use XAutoRepeatOff.

XAutoRepeatOff(display)
    Display *display;

display  Specifies the connection to the X server.

The XAutoRepeatOff function turns off auto-repeat for the keyboard on the specified display.

To ring the bell, use XBell.

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

display  Specifies the connection to the X server.
percent  Specifies the volume for the bell, which can range from -100 to 100 inclusive.

The XBell function rings the bell on the keyboard on the specified display, if possible. The specified volume is relative to the base volume for the keyboard. If the value for the percent argument is not in the range -100 to 100 inclusive, a BadValue error results. The volume at which the bell rings when the percent argument is nonnegative is:

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

The volume at which the bell rings when the percent argument is negative is:

    base + [(base * percent) / 100]

To change the base volume of the bell, use XChangeKeyboardControl.

XBell can generate a BadValue error.
To obtain a bit vector that describes the state of the keyboard, use `XQueryKeymap`.

```c
XQueryKeymap(display, keys_return)
    Display *display;
    char keys_return[32];
```

- **display**: Specifies the connection to the X server.
- **keys_return**: Returns an array of bytes that identifies which keys are pressed down. Each bit represents one key of the keyboard.

The `XQueryKeymap` function returns a bit vector for the logical state of the keyboard, where each bit set to 1 indicates that the corresponding key is currently pressed down. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7 with the least-significant bit in the byte representing key 8N.

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.

To set the mapping of the pointer buttons, use `XSetPointerMapping`.

```c
int XSetPointerMapping(display, map, nmap)
    Display *display;
    unsigned char map[];
    int nmap;
```

- **display**: Specifies the connection to the X server.
- **map**: Specifies the mapping list.
- **nmap**: Specifies the number of items in the mapping list.

The `XSetPointerMapping` function sets the mapping of the pointer. If it succeeds, the X server generates a MappingNotify event, and `XSetPointerMapping` returns MappingSuccess. Element map[i] defines the logical button number for the physical button i+1. The length of the list must be the same as `XGetPointerMapping` would return, or a BadValue error results. A zero element disables a button, and elements are not restricted in value by the number of physical buttons. However, no two elements can have the same nonzero value, or a BadValue error results. If any of the buttons to be altered are logically in the down state, `XSetPointerMapping` returns MappingBusy, and the mapping is not changed.

`XSetPointerMapping` can generate a BadValue error.

To get the pointer mapping, use `XGetPointerMapping`.

```c
int XGetPointerMapping(display, map_return, nmap)
    Display *display;
    unsigned char map_return[];
    int nmap;
```

- **display**: Specifies the connection to the X server.
- **map_return**: Returns the mapping list.
- **nmap**: Specifies the number of items in the mapping list.

The `XGetPointerMapping` function returns the current mapping of the pointer. Pointer buttons are numbered starting from one. `XGetPointerMapping` returns the number of physical buttons actually on the pointer. The nominal mapping for a pointer is map[i]=i+1. The nmap argument specifies the length of the array where the pointer mapping is returned, and only the first nmap elements are returned in map_return.

To control the pointer's interactive feel, use `XChangePointerControl`.
XChangePointerControl(display, do_accel, do_threshold, accel_numerator, accel_denominator, threshold)

Display *display;
Bool do_accel, do_threshold;
int accel_numerator, accel_denominator;
int threshold;

display Specifies the connection to the X server.
do_accel Specifies a Boolean value that controls whether the values for the accel_numerator or accel_denominator are used.
do_threshold Specifies a Boolean value that controls whether the value for the threshold is used.

accel_numerator Specifies the numerator for the acceleration multiplier.
accel_denominator Specifies the denominator for the acceleration multiplier.
threshold Specifies the acceleration threshold.

The XChangePointerControl function defines how the pointing device moves. The acceleration, expressed as a fraction, is a multiplier for movement. For example, specifying 3/1 means the pointer moves three times as fast as normal. The fraction may be rounded arbitrarily by the X server. Acceleration only takes effect if the pointer moves more than threshold pixels at once and only applies to the amount beyond the value in the threshold argument. Setting a value to -1 restores the default. The values of the do_accel and do_threshold arguments must be True for the pointer values to be set, or the parameters are unchanged. Negative values (other than -1) generate a BadValue error, as does a zero value for the accel_denominator argument.

XChangePointerControl can generate a BadValue error.

To get the current pointer parameters, use XGetPointerControl.

XGetPointerControl(display, accel_numerator_return, accel_denominator_return, threshold_return)

Display *display;
int *accel_numerator_return, *accel_denominator_return;
int *threshold_return;

display Specifies the connection to the X server.

accel_numerator_return Returns the numerator for the acceleration multiplier.
accel_denominator_return Returns the denominator for the acceleration multiplier.
threshold_return Returns the acceleration threshold.

The XGetPointerControl function returns the pointer's current acceleration multiplier and acceleration threshold.

12.7. Keyboard Encoding

A KeyCode represents a physical (or logical) key. KeyCodes lie in the inclusive range [8, 255]. A KeyCode value carries no intrinsic information, although server implementors may attempt to encode geometry (for example, matrix) information in some fashion so that it can be interpreted in a server-dependent fashion. The mapping between keys and KeyCodes cannot be changed.

A KeySym is an encoding of a symbol on the cap of a key. The set of defined KeySyms includes the ISO Latin character sets (1-4), Katakana, Arabic, Cyrillic, Greek, Technical,
Special, Publishing, APL, Hebrew, and a special miscellany of keys found on keyboards (Return, Help, Tab, and so on). To the extent possible, these sets are derived from international standards. In areas where no standards exist, some of these sets are derived from Digital Equipment Corporation standards. The list of defined symbols can be found in <X11/keysymdef.h>. Unfortunately, some C preprocessors have limits on the number of defined symbols. If you must use KeySyms not in the Latin 1–4, Greek, and miscellaneous classes, you may have to define a symbol for those sets. Most applications usually only include <X11/keysym.h>, which defines symbols for ISO Latin 1–4, Greek, and miscellaneous.

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 the 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. Which group to use is determined by the 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, the modifier state also determines which KeySym to use. The first KeySym is used when the Shift and Lock modifiers are off. The second KeySym is used when the Shift modifier is on, 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.

No spatial geometry of the symbols on the key is defined by their order in the KeySym list, although a geometry might be defined on a vendor-specific basis. The X server does not use the mapping between KeyCodes and KeySyms. Rather, it stores it merely for reading and writing by clients.

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.

To obtain the legal KeyCodes for a display, use XDisplayKeycodes.

XDisplayKeycodes(display, min_keycodes_return, max_keycodes_return)
    Display *display;
    int *min_keycodes_return, *max_keycodes_return;
display Specifies the connection to the X server.

min_keycodes_return
Returns the minimum number of KeyCodes.

max_keycodes_return
Returns the maximum number of KeyCodes.

The XDisplayKeycodes function returns the min-keycodes and max-keycodes supported by the specified display. The minimum number of KeyCodes returned is never less than 8, and the maximum number of KeyCodes returned is never greater than 255. Not all KeyCodes in this range are required to have corresponding keys.

To obtain the symbols for the specified KeyCodes, use XGetKeyboardMapping.

KeySym *XGetKeyboardMapping(display, first_keycode, keycode_count, keysyms_per_keycode_return)

Display *display;
KeyCode first_keycode;
int keycode_count;
int *keysyms_per_keycode_return;

display Specifies the connection to the X server.

first_keycode Specifies the first KeyCode that is to be returned.

keycode_count Specifies the number of KeyCodes that are to be returned.

keysyms_per_keycode_return Returns the number of KeySyms per KeyCode.

The XGetKeyboardMapping function returns the symbols for the specified number of KeyCodes starting with first_keycode. The value specified in first_keycode must be greater than or equal to min_keycode as returned by XDisplayKeycodes, or a BadValue error results. In addition, the following expression must be less than or equal to max_keycode as returned by XDisplayKeycodes:

first_keycode + keycode_count - 1

If this is not the case, a BadValue error results. The number of elements in the KeySyms list is:

keycode_count * keysyms_per_keycode_return

KeySym number N, counting from zero, for KeyCode K has the following index in the list, counting from zero:

(K - first_code) * keysyms_per_code_return + N

The X server arbitrarily chooses the keysyms_per_keycode_return value to be large enough to report all requested symbols. A special KeySym value of NoSymbol is used to fill in unused elements for individual KeyCodes. To free the storage returned by XGetKeyboardMapping, use XFree.

XGetKeyboardMapping can generate a BadValue error.

To change the keyboard mapping, use XChangeKeyboardMapping.
XChangeKeyboardMapping(display, first_keycode, keysyms_per_keycode, keysyms, num_codes)
          Display *display;
          int first_keycode;
          int keysyms_per_keycode;
          KeySym *keysyms;
          int num_codes;

display  Specifies the connection to the X server.
first_keycode  Specifies the first KeyCode that is to be changed.
keysyms_per_keycode  Specifies the number of KeySyms per KeyCode.
keysyms  Specifies an array of KeySyms.
num_codes  Specifies the number of KeyCodes that are to be changed.

The XChangeKeyboardMapping function defines the symbols for the specified number of
KeyCodes starting with first_keycode. The symbols for KeyCodes outside this range remain
unchanged. The number of elements in keysyms must be:

num_codes * keysyms_per_keycode

The specified first_keycode must be greater than or equal to min_keycode returned by
XDisplayKeycodes, or a BadValue error results. In addition, the following expression must
be less than or equal to max_keycode as returned by XDisplayKeycodes, or a BadValue
error results:

first_keycode + num_codes - 1

KeySym number N, counting from zero, for KeyCode K has the following index in keysyms,
counting from zero:

(K - first_keycode) * keysyms_per_keycode + N

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

There is no requirement that the X server interpret this mapping. It is merely stored for reading
and writing by clients.

XChangeKeyboardMapping can generate BadAlloc and BadValue errors.

The next four functions make use of the XModifierKeymap data structure, which contains:
typedef struct {
    int max_keys_per_mod;  /* This server's max number of keys per modifier */
    KeyCode *modifiermap;  /* An 8 by max_keys_per_mod array of the modifiers */
} XModifierKeymap;

To create an XModifierKeymap structure, use XNewModifiermap.

XModifierKeymap *XNewModifiermap(max_keys_per_mod)
int max_keys_per_mod;

max_keys_per_mod  Specifies the number of KeyCode entries preallocated to the modifiers in the
map.

The XNewModifiermap function returns a pointer to XModifierKeymap structure for later
use.
To add a new entry to an XModifierKeymap structure, use XInsertModifiermapEntry.

```c
XModifierKeymap *XInsertModifiermapEntry(modmap, keycode_entry, modifier)

    XModifierKeymap *modmap;
    KeyCode keycode_entry;
    int modifier;
```

- `modmap` Specifies the XModifierKeymap structure.
- `keycode_entry` Specifies the KeyCode.
- `modifier` Specifies the modifier.

The XInsertModifiermapEntry function adds the specified KeyCode to the set that controls the specified modifier and returns the resulting XModifierKeymap structure (expanded as needed).

To delete an entry from an XModifierKeymap structure, use XDeleteModifiermapEntry.

```c
XModifierKeymap *XDeleteModifiermapEntry(modmap, keycode_entry, modifier)

    XModifierKeymap *modmap;
    KeyCode keycode_entry;
    int modifier;
```

- `modmap` Specifies the XModifierKeymap structure.
- `keycode_entry` Specifies the KeyCode.
- `modifier` Specifies the modifier.

The XDeleteModifiermapEntry function deletes the specified KeyCode from the set that controls the specified modifier and returns a pointer to the resulting XModifierKeymap structure.

To destroy an XModifierKeymap structure, use XFreeModifiermap.

```c
XFreeModifiermap(modmap)
```

- `modmap` Specifies the XModifierKeymap structure.

The XFreeModifiermap function frees the specified XModifierKeymap structure.

To set the KeyCodes to be used as modifiers, use XSetModifierMapping.

```c
int XSetModifierMapping(display, modmap)

    Display *display;
    XModifierKeymap *modmap;
```

- `display` Specifies the connection to the X server.
- `modmap` Specifies the XModifierKeymap structure.

The XSetModifierMapping function specifies the KeyCodes of the keys (if any) that are to be used as modifiers. If it succeeds, the X server generates a MappingNotify event, and XSetModifierMapping returns MappingSuccess. X permits at most eight modifier keys. If more than eight are specified in the XModifierKeymap structure, a BadLength error results.

The modifiermap member of the XModifierKeymap structure contains eight sets of max_keypermod KeyCodes, one for each modifier in the order Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, and Mod5. Only nonzero KeyCodes have meaning in each set, and zero KeyCodes are ignored. In addition, all of the nonzero KeyCodes must be in the range specified by min_keycode and max_keycode in the Display structure, or a BadValue error results.

An X 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 modifier keys are not supported. If some such restriction is violated, the status reply is MappingFailed, and none of the modifiers are changed. If the new KeyCodes specified for a modifier differ from those currently defined and any (current or new) keys for that modifier are in the logically down state, XSetModifierMapping returns MappingBusy, and none of the modifiers is changed.

XSetModifierMapping can generate BadAlloc and BadValue errors.

To obtain the KeyCodes used as modifiers, use XGetModifierMapping.

XModifierKeymap *XGetModifierMapping(display)
   Display *display;

*display Specifies the connection to the X server.

The XGetModifierMapping function returns a pointer to a newly created XModifierKeymap structure that contains the keys being used as modifiers. The structure should be freed after use by calling XFreeModifiermap. If only zero values appear in the set for any modifier, that modifier is disabled.
Chapter 13
Locales and Internationalized Text Functions

An internationalized application is one which is adaptable to the requirements of different native languages, local customs, and character string encodings. The process of adapting the operation to a particular native language, local custom, or string encoding is called localization. A goal of internationalization is to permit localization without program source modifications or recompilation.

Internationalization in X is based on the concept of a locale. A locale defines the "localized" behavior of a program at run-time. Locales affect Xlib in its:

- Encoding and processing of input method text
- Encoding of resource files and values
- Encoding and imaging of text strings
- Encoding and decoding for inter-client text communication

Characters from various languages are represented in a computer using an encoding. Different languages have different encodings, and there are even different encodings for the same characters in the same language.

This chapter defines support for localized text imaging and text input, and the locale mechanism which controls all locale-dependent Xlib functions. Sets of functions are provided for multibyte (char *) text as well as wide character (wchar_t) text in the form supported by the host C language environment. The multibyte and wide character functions are equivalent except for the form of the text argument.

The Xlib internationalization functions are not meant to provide support for multilingual applications (mixing multiple languages within a single piece of text), but they make it possible to implement applications that work in limited fashion with more than one language in independent contexts.

13.1. X Locale Management

X supports a one or more of the locales defined by the host environment. On implementations that conform to the ANSI C library, the locale announcement method is setlocale. This function configures the locale operation of both the host C library and Xlib. The operation of Xlib is governed by the LC_CTYPE category; this is called the current locale. An implementation is permitted to provide implementation-dependent mechanisms for announcing the locale in addition to setlocale.

On implementations that do not conform to the ANSI C library, the locale announcement method is Xlib implementation-dependent.

The mechanism by which the semantic operation of Xlib is defined for a specific locale is implementation-dependent.

X is not required to support all the locales supported by the host. To determine if the current locale is supported by X, use XSupportsLocale.

Bool XSupportsLocale()

The XSupportsLocale function returns True if Xlib functions are capable of operating under the current locale. If it returns False, Xlib locale-dependent functions for which the XLocaleNotSupported return status is defined will return XLocaleNotSupported. Other Xlib locale-dependent routines will operate in the "C" locale.

222
The client is responsible for selecting its locale and X modifiers. Clients should provide a means for the user to override the clients’ locale selection at client invocation. Most single-display X clients operate in a single locale for both X and the host processing environment. They will configure the locale by calling three functions: the host locale configuration function, XSupportsLocale, and XSetLocaleModifiers.

The semantics of certain categories of X internationalization capabilities can be configured by setting modifiers. Modifiers are named by implementation-dependent and locale-specific strings. The only standard use for this capability at present is selecting one of several styles of keyboard input method.

To configure Xlib locale modifiers for the current locale, use XSetLocaleModifiers.

```c
char *XSetLocaleModifiers(char *modifier_list)
```

`modifier_list` Specifies the modifiers.

XSetLocaleModifiers sets the X modifiers for the current locale setting. The modifier_list argument is a null-terminated string of the form "@category=value", that is, having zero or more concatenated "@category=value" entries where category is a category name and value is the (possibly empty) setting for that category. The values are encoded in the current locale. Category names are restricted to the POSIX Portable Filename Character Set.

The local host X locale modifiers announcer (on POSIX-compliant systems, the XMODIFIERS environment variable) is appended to the modifier_list to provide default values on the local host. If a given category appears more than once in the list, the first setting in the list is used. If a given category is not included in the full modifier list, the category is set to an implementation-dependent default for the current locale. An empty value for a category explicitly specifies the implementation-dependent default.

If the function is successful, it returns a pointer to a string. The contents of the string are such that a subsequent call with that string (in the same locale) will restore the modifiers to the same settings. If modifier_list is a NULL pointer, XSetLocaleModifiers also returns a pointer to such a string, and the current locale modifiers are not changed.

If invalid values are given for one or more modifier categories supported by the locale, a NULL pointer is returned, and none of the current modifiers are changed.

At program startup the modifiers that are in effect are unspecified until the first successful call to set them. Whenever the locale is changed, the modifiers that are in effect become unspecified until the next successful call to set them. Clients should always call XSetLocaleModifiers with a non-NULL modifier_list after setting the locale, before they call any locale-dependent Xlib routine.

The only standard modifier category currently defined is "im", which identifies the desired input method. The values for input method are not standardized. A single locale may use multiple input methods, switching input method under user control. The modifier may specify the initial input method in effect, or an ordered list of input methods. Multiple input methods may be specified in a single im value string in an implementation-dependent manner.

The returned modifiers string is owned by Xlib and should not be modified or freed by the client. It may be freed by Xlib after the current locale or modifiers is changed. Until freed, it will not be modified by Xlib.

The recommended procedure for clients initializing their locale and modifiers is to obtain locale and modifier announcers separately from one of the following prioritized sources:

- A command line option
- A resource
- The empty string ("")
The first of these that is defined should be used. Note that when a locale command line option or locale resource is defined, the effect should be to set all categories to the specified locale, overriding any category-specific settings in the local host environment.

### 13.2. Locale and Modifier Dependencies

The internationalized Xlib functions operate in the current locale configured by the host environment and X locale modifiers set by `XSetLocaleModifiers`, or in the locale and modifiers configured at the time some object supplied to the function was created. For each locale-dependent function, the following table describes the locale (and modifiers) dependency:

<table>
<thead>
<tr>
<th>locale from...</th>
<th>Affects the function...</th>
<th>in the...</th>
</tr>
</thead>
<tbody>
<tr>
<td>setlocale</td>
<td>XSupportsLocale</td>
<td>locale queried</td>
</tr>
<tr>
<td></td>
<td>XSetLocaleModifiers</td>
<td>locale modified</td>
</tr>
<tr>
<td>Resources:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setlocale</td>
<td>XrmGetFileDatabase</td>
<td>locale of XrmDatabase</td>
</tr>
<tr>
<td></td>
<td>XrmGetStringDatabase</td>
<td></td>
</tr>
<tr>
<td>XrmDatabase</td>
<td>XrmPutFileDatabase</td>
<td>locale of XrmDatabase</td>
</tr>
<tr>
<td></td>
<td>XrmLocaleOfDatabase</td>
<td></td>
</tr>
<tr>
<td>Setting Standard Properties:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setlocale</td>
<td>XmbSetWMProperties</td>
<td>encoding of supplied/returned text (some WM property text in environment locale)</td>
</tr>
<tr>
<td></td>
<td>XwcTextPropertyToTextList</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XmbTextPropertyToTextList</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XwcTextListToTextProperty</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XmbTextListToTextProperty</td>
<td></td>
</tr>
<tr>
<td>Text Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setlocale</td>
<td>XOpenIM</td>
<td>XIM input method selection</td>
</tr>
<tr>
<td>XIM</td>
<td>XCreateIC</td>
<td>XIC input method configuration</td>
</tr>
<tr>
<td></td>
<td>XLocaleOfIM, etc.</td>
<td>queried locale</td>
</tr>
<tr>
<td>XIC</td>
<td>XmbLookupText</td>
<td>keyboard layout, encoding of returned text</td>
</tr>
<tr>
<td></td>
<td>XwcLookupText</td>
<td></td>
</tr>
<tr>
<td>Text Drawing:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setlocale</td>
<td>XCreateFontSet</td>
<td>charsets of fonts in XFontSet</td>
</tr>
<tr>
<td>XFontSet</td>
<td>XmbDrawText,</td>
<td>locale of supplied text, locale of supplied text, locale-dependent metrics</td>
</tr>
<tr>
<td></td>
<td>XwcDrawText, etc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XExtentsOfFontSet, etc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XmbTextExtents,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XwcTextExtents, etc.</td>
<td></td>
</tr>
<tr>
<td>Xlib Errors:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setlocale</td>
<td>XGetErrorDatabaseText</td>
<td>locale of error message</td>
</tr>
<tr>
<td></td>
<td>XGetErrorText</td>
<td></td>
</tr>
</tbody>
</table>
All text strings processed by internationalized Xlib functions are assumed to begin in the initial state of the encoding of the locale, if the encoding is state-dependent. All Xlib routines behave as if they do not change the current locale or X modifier setting. (This means that if they do change locale or call XSetLocaleModifiers with a non-NULL argument, they must save and restore the current state on entry and exit.) Also, Xlib functions on implementations that conform to the ANSI C library do not alter the global state associated with the ANSI C functions mblen, mbtowc, wctomb, and strtok.

### 13.3. Creating and Freeing a Font Set

Xlib international text drawing is done using a set of one or more fonts, as needed for the locale of the text. Fonts are loaded according to a list of base font names supplied by the client, and the charsets required by the locale. The XFontSet is an opaque type.

To create an international text drawing font set, use `XCreateFontSet`.

```
XFontSet XCreateFontSet(display, base_font_name_list, missing_charset_list_return,
            missing_charset_count_return, def_string_return);
```

- `display`: Specifies the connection to the X server.
- `base_font_name_list`: Specifies the base font names.
- `missing_charset_list_return`: Returns the missing charsets.
- `missing_charset_count_return`: Returns the number of missing charsets.
- `def_string_return`: Returns the string drawn for missing charsets.

The `XCreateFontSet` function creates a font set for the specified display. The font set is bound to the current locale when `XCreateFontSet` is called. The font_set may be used in subsequent calls to obtain font and character information, and to image text in the locale of the font_set.

The `base_font_name_list` argument is a list of base font names which Xlib uses to load the fonts needed for the locale. The base font names are a comma-separated list. The string is null-terminated, and is assumed to be in the Host Portable Character Encoding; otherwise, the result is implementation dependent. Whitespace immediately on either side of a separating comma is ignored.

Use of XLFD font names permits Xlib to obtain the fonts needed for a variety of locales from a single locale-independent base font name. The single base font name should name a family of fonts whose members are encoded in the various charsets needed by the locales of interest.

An XLFD base font name can explicitly name a charset needed for the locale. This allows the user to specify an exact font for use with a charset required by a locale, fully controlling the font selection.
If a base font name is not an XLFD name, Xlib will attempt to obtain an XLFD name from the font properties for the font. If this action is successful in obtaining an XLFD name, the XBaseFontNameListOfFontSet function will return this XLFD name instead of the client-supplied name.

The following algorithm is used to select the fonts that will be used to display text with the XFontSet:

For each font charset required by the locale, the base font name list is searched for the first one of the following cases that names a set of fonts that exist at the server:

1. The first XLFD-conforming base font name that specifies the required charset or a superset of the required charset in its CharSetRegistry and CharSetEncoding fields. The implementation may use a base font name whose specified charset is a superset of the required charset, for example, an ISO8859-1 font for an ASCII charset.
2. The first set of one or more XLFD-conforming base font names that specify one or more charsets that can be remapped to support the required charset. The Xlib implementation may recognize various mappings from a required charset to one or more other charsets, and use the fonts for those charsets. For example, JIS Roman is ASCII with tilde and backslash replaced by yen and overbar; Xlib may load an ISO8859-1 font to support this character set, if a JIS Roman font is not available.
3. The first XLFD-conforming font name, or the first non-XLFD font name for which an XLFD font name can be obtained, combined with the required charset (replacing the CharSetRegistry and CharSetEncoding fields in the XLFD font name). As in case 1, the implementation may use a charset which is a superset of the required charset.
4. The first font name that can be mapped in some implementation-dependent manner to one or more fonts that support imaging text in the charset.

For example, assume a locale required the charsets:

ISO8859-1
JISX0208.1983
JISX0201.1976
GB2312-1980.0

The user could supply a base_font_name_list which explicitly specifies the charsets, insuring that specific fonts get used if they exist:


Or he could supply a base_font_name_list which omits the charsets, letting Xlib select font charsets required for the locale:

"-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-240,\n-JIS-Fixed-Medium-R-Normal--26-180-100-100-C-120,\n-GB-Fixed-Medium-R-Normal--26-180-100-100-C-240,\n-Adobe-Courier-Bold-R-Normal--25-180-100-100-M-150"

Or he could simply supply a single base font name which allows Xlib to select from all available fonts which meet certain minimum XLFD property requirements:

"-.*-.*-R-Normal-*.180-100-100.-.*"

If XCreateFontSet is unable to create the font set, either because there is insufficient memory or because the current locale is not supported, XCreateFontSet returns NULL, missing_charset_list_return is set to NULL, and missing_charset_count_return is set to zero. If fonts exist for all of the charsets required by the current locale, XCreateFontSet returns a
valid XFontSet, missing_charset_list_return is set to NULL, and missing_charset_count_return is set to zero.

If no font exists for one or more of the required charsets, XCreateFontSet sets missing_charset_list_return to a list of one or more null-terminated charset names for which no font exists, and sets missing_charset_count_return to the number of missing fonts. The charsets are from the list of the required charsets for the encoding of the locale, and do not include any charsets to which Xlib may be able to remap a required charset.

If no font exists for any of the required charsets, or if the locale definition in Xlib requires that a font exist for a particular charset and a font is not found for that charset, XCreateFontSet returns NULL. Otherwise, XCreateFontSet returns a valid XFontSet to font_set.

When an Xmb/wc drawing or measuring function is called with an XFontSet that has missing charsets, some characters in the locale will not be drawable. If def_string_return is non-NUL1, XCreateFontSet returns a pointer to a string which represents the glyph(s) which are drawn with this XFontSet when the charsets of the available fonts do not include all font glyph(s) required to draw a codepoint. The string does not necessarily consist of valid characters in the current locale and is not necessarily drawn with the fonts loaded for the font set, but the client can draw and measure the "default glyphs" by including this string in a string being drawn or measured with the XFontSet.

If the string returned to def_string_return is the empty string (""), no glyphs are drawn, and the escapement is zero. The returned string is null-terminated. It is owned by Xlib and should not be modified or freed by the client. It will be freed by a call to XFreeFontSet with the associated XFontSet. Until freed, its contents will not be modified by Xlib.

The client is responsible for constructing an error message from the missing charset and default string information, and may choose to continue operation in the case that some fonts did not exist.

The returned XFontSet and missing charset list should be freed with XFreeFontSet and XFreeStringList, respectively. The client-supplied base_font_name_list may be freed by the client after calling XCreateFontSet.

To obtains a list of XFontStruct structures and full font names given an XFontSet, use XFontsOfFontSet.

int XFontsOfFontSet(font_set, font_struct_list_return, font_name_list_return)

XFontSet font_set;
XFontStruct ***font_struct_list_return;
char ***font_name_list_return;

font_set Specifies the font set.
font_struct_list_return
Returns the list of font structs.
font_name_list_return
Returns the list of font names.

The XFontsOfFontSet function returns a list of one or more XFontStructs and font names for the fonts used by the Xmb and Xwc layers, for the given font set. A list of pointers to the XFontStruct structures is returned to font_struct_list_return. A list of pointers to null-terminated fully specified font name strings in the locale of the font set is returned to font_name_list_return. The font_name_list order corresponds to the font_struct_list order. The number of XFontStruct structures and font names is returned as the value of the function.

Because it is not guaranteed that a given character will be imaged using a single font glyph, there is no provision for mapping a character or default string to the font properties, font ID, or direction hint for the font for the character. The client may access the XFontStruct list to obtain these values for all the fonts currently in use.
It is not required that fonts be loaded from the server at the creation of an XFontSet. Xlib may choose to cache font data, loading it only as needed to draw text or compute text dimensions. Therefore, existence of the per_char metrics in the XFontStruct structures in the XFontStructSet is undefined. Also, note that all properties in the XFontStruct structures are in the STRING encoding.

The XFontStruct and font name lists are owned by Xlib and should not be modified or freed by the client. They will be freed by a call to XFreeFontSet with the associated XFontSet. Until freed, its contents will not be modified by Xlib.

To obtain the base font name list and the selected font name list given an XFontSet, use XBaseFontNameListOfFontSet.

```c
char *XBaseFontNameListOfFontSet(font_set)
    XFontSet font_set;
```

`font_set` Specifies the font set.

The XBaseFontNameListOfFontSet function returns the original base font name list supplied by the client when the XFontSet was created. A null-terminated string containing a list of comma-separated font names is returned as the value of the function. Whitespace may appear immediately on either side of separating commas.

If XCreateFontSet obtained an XLFD name from the font properties for the font specified by a non-XLFD base name, the XBaseFontNameListOfFontSet function will return the XLFD name instead of the non-XLFD base name.

The base font name list is owned by Xlib and should not be modified or freed by the client. It will be freed by a call to XFreeFontSet with the associated XFontSet. Until freed, its contents will not be modified by Xlib.

To obtain the locale name given an XFontSet, use XLocaleOfFontSet.

```c
char *XLocaleOfFontSet(font_set)
    XFontSet font_set;
```

`font_set` Specifies the font set.

The XLocaleOfFontSet function returns the name of the locale bound to the specified XFontSet, as a null-terminated string.

The returned locale name string is owned by Xlib and should not be modified or freed by the client. It may be freed by a call to XFreeFontSet with the associated XFontSet. Until freed, it will not be modified by Xlib.

To free a font set, use XFreeFontSet.

```c
void XFreeFontSet(display, font_set)
    Display *display;
    XFontSet font_set;
```

`display` Specifies the connection to the X server.

`font_set` Specifies the font set.

The XFreeFontSet function frees the specified font set. The associated base font name list, font name list, XFontStruct list, and XFontSetExtents, if any, are freed.

13.4. Obtaining Font Set Metrics

Metrics for the internationalized text drawing functions are defined in terms of a primary draw direction, which is the default direction in which the character origin advances for each succeeding character in the string. The Xlib interface is currently defined to support only a
left-to-right primary draw direction. The drawing origin is the position passed to the drawing function when the text is drawn. The baseline is a line drawn through the drawing origin parallel to the primary draw direction. Character ink is the pixels painted in the foreground color and does not include interline or intercharacter spacing or image text background pixels.

The drawing functions are allowed to implement implicit text directionality control, reversing the order in which characters are rendered along the primary draw direction in response to locale-specific lexical analysis of the string.

Regardless of the character rendering order, the origins of all characters are on the primary draw direction side of the drawing origin. The screen location of a particular character image may be determined with XmbTextPerCharExtents or XwcTextPerCharExtents.

The drawing functions are allowed to implement context-dependent rendering, where the glyphs drawn for a string are not simply a concatenation of the glyphs that represent each individual character. A string of two characters drawn with XmbDrawString may render differently than if the two characters were drawn with separate calls to XmbDrawString. If the client appends or inserts a character in a previously drawn string, the client may need to redraw some adjacent characters in order to obtain proper rendering.

To find out about context-dependent rendering, use XContextDependentDrawing.

```c
Bool XContextDependentDrawing/font_set /
  XFontSet font_set;
```

`font_set` Specifies the font set.

The XContextDependentDrawing function returns True if text drawn with the font_set might include context-dependent drawing.

The drawing functions do not interpret newline, tab, or other control characters. The behavior when non-printing characters other than space are drawn is implementation-dependent. It is the client’s responsibility to interpret control characters in a text stream.

The maximum character extents for the fonts that are used by the text drawing layers may be accessed by the XFontSetExtents structure:

```c
typedef struct {
  XRectangle max_ink_extent; /* over all drawable characters */
  XRectangle max.logical_extent; /* over all drawable characters */
} XFontSetExtents;
```

The XRectangles used to return font set metrics are the usual Xlib screen-oriented XRectangles, with x, y giving the upper left corner, and width and height always positive.

The max_ink_extent member gives the maximum extent, over all drawable characters, of the rectangles which bound the character glyph image drawn in the foreground color, relative to a constant origin. See XmbTextExtents and XwcTextExtents for detailed semantics.

The max_logical_extent member gives the maximum extent, over all drawable characters, of the rectangles which specify minimum spacing to other graphical features, relative to a constant origin. Other graphical features drawn by the client, for example, a border surrounding the text, should not intersect this rectangle. The max_logical_extent member should be used to compute minimum inter-line spacing and the minimum area which must be allowed in a text field to draw a given number of arbitrary characters.

Due to context-dependent rendering, appending a given character to a string may increase the string’s extent by an amount which exceeds the font’s max extent:

```
max possible added extent = (max.extent * <total # chars>). - prev_string.extent
```

The rectangles for a given character in a string can be obtained from XmbPerCharExtents or XwcPerCharExtents.
To obtain the maximum extents structure given an XFontSet, use XExtentsOfFontSet.

XFontSetExtents *XExtentsOfFontSet(font_set)
    XFontSet font_set;

font_set    Specifies the font set.

The XExtentsOfFontSet function returns an XFontSetExtents structure for the fonts used by the Xmb and Xwc layers, for the given font set.

The XFontSetExtents structure is owned by Xlib and should not be modified or freed by the client. It will be freed by a call to XFreeFontSet with the associated XFontSet. Until freed, its contents will not be modified by Xlib.

To obtain the escapement in pixels of the specified text as a value, use XmbTextEscapement or XwcTextEscapement.

int XmbTextEscapement(font_set, string, num_bytes)
    XFontSet font_set;
    char *string;
    int num_bytes;

int XwcTextEscapement(font_set, string, num_wchars)
    XFontSet font_set;
    wchar_t *string;
    int num_wchars;

font_set    Specifies the font set.
string    Specifies the character string.
num_bytes    Specifies the number of bytes in the string argument.
num_wchars    Specifies the number of characters in the string argument.

The XmbTextEscapement and XwcTextEscapement functions return the escapement in pixels of the specified string as a value, using the fonts loaded for the specified font set. The escapement is the distance in pixels in the primary draw direction from the drawing origin to the origin of the next character to be drawn, assuming that the rendering of the next character is not dependent on the supplied string.

Regardless of the character rendering order, the escapement is always positive.

To obtain the overall_ink_return and overall_logical_return arguments, the overall bounding box of the string's image, and a logical bounding box, use XmbTextExtents or XwcTextExtents.

int XmbTextExtents(font_set, string, num_bytes, overall_return)
    XFontSet font_set;
    char *string;
    int num_bytes;
    XRectangle *overall_ink_return;
    XRectangle *overall_logical_return;

int XwcTextExtents(font_set, string, num_wchars, overall_return)
    XFontSet font_set;
    wchar_t *string;
    int num_wchars;
    XRectangle *overall_ink_return;
    XRectangle *overall_logical_return;

font_set    Specifies the font set.
string  Specifies the character string.
num_bytes  Specifies the number of bytes in the string argument.
num_wchars  Specifies the number of characters in the string argument.
overall_ink_return  Returns the overall ink dimensions.
overall_logical_return  Returns the overall logical dimensions.

The XmbTextExtents and XwcTextExtents functions set the components of the specified overall_ink_return and overall_logical_return arguments to the overall bounding box of the string's image, and a logical bounding box for spacing purposes, respectively. They return the value returned by XmbTextEscapement or XwcTextEscapement. These metrics are relative to the drawing origin of the string, using the fonts loaded for the specified font set.

If the overall_ink_return argument is non-NULL, it is set to the bounding box of the string's character ink. Note that the overall_ink_return for a non-descending horizontally drawn Latin character is conventionally entirely above the baseline, that is, overall_ink_return.height <= -overall_ink_return.y. The overall_ink_return for a nonkemed character is entirely at and to the right of the origin, that is, overall_ink_return.x >= 0. A character consisting of a single pixel at the origin would set overall_ink_return fields y = 0, x = 0, width = 1, height = 1.

If the overall_logical_return argument is non-NULL, it is set to the bounding box which provides minimum spacing to other graphical features for the string. Other graphical features, for example, a border surrounding the text, should not intersect this rectangle.

When the XFonSet has missing charsets, metrics for each unavailable character are taken from the default string returned by XCreateFontSet so that the metrics represent the text as it will actually be drawn. The behavior for an invalid codepoint is undefined.

To determine the effective drawing origin for a character in a drawn string, the client should call XmbTextPerCharExtents on the entire string, then on the character, and subtract the x values of the returned XRectangles for the character. This is useful to redraw portions of a line of text or to justify words, but for context-dependent rendering the client should not assume that it can redraw the character by itself and get the same rendering.

To obtain per-character information for a text string, use XmbTextPerCharExtents or XwcTextPerCharExtents.

Status XmbTextPerCharExtents(font_set, string, num_bytes, ink_array_return, logical_array_return, array_size, num_chars_return, overall_return)
    XFontSet *font_set;
    char *string;
    int num_bytes;
    XRectangle *ink_array_return;
    XRectangle *logical_array_return;
    int array_size;
    int *num_chars_return;
    XRectangle *overall_ink_return;
    XRectangle *overall_logical_return;
Status XwcTextPerCharExtents(font_set, string, num_wchars, ink_array_return, logical_array_return, array_size, num_chars_return, overall_return)

XFontSet font_set;
wchar_t *string;
int num_wchars;
XRectangle *ink_array_return;
XRectangle *logical_array_return;
int array_size;
int *num_chars_return;
XRectangle *overall_ink_return;
XRectangle *overall_logical_return;

font_set Specifies the font set.
string Specifies the character string.
um_bytes Specifies the number of bytes in the string argument.
um_wchars Specifies the number of characters in the string argument.
ink_array_return Returns the ink dimensions for each character.
logical_array_return Returns the logical dimensions for each character.
array_size Specifies the size of ink_array_return and logical_array_return. Note that the caller must pass in arrays of this size.
um_chars_return Returns the number characters in the string argument.
overall_ink_return Returns the overall ink extents of the entire string.
overall_logical_return Returns the overall logical extents of the entire string.

The XmbTextPerCharExtents and XwcTextPerCharExtents return the text dimensions of each character of the specified text, using the fonts loaded for the specified font set. Each successive element of ink_array_return and logical_array_return is set to the successive character’s drawn metrics, relative to the drawing origin of the string, one XRectangle for each character in the supplied text string. The number of elements of ink_array_return and logical_array_return that have been set is returned to num_chars_return.

Each element of ink_array_return is set to the bounding box of the corresponding character’s drawn foreground color. Each element of logical_array_return is set to the bounding box which provides minimum spacing to other graphical features for the corresponding character. Other graphical features should not intersect any of the logical_array_return rectangles.

Note that an XRectangle represents the effective drawing dimensions of the character, regardless of the number of font glyphs that are used to draw the character, or the direction in which the character is drawn. If multiple characters map to a single character glyph, the dimensions of all the XRectangles of those characters are the same.

When the XFontSet has missing charsets, metrics for each unavailable character are taken from the default string returned by XCreateFontSet, so that the metrics represent the text as it will actually be drawn. The behavior for an invalid codepoint is undefined.

If the array_size is too small for the number of characters in the supplied text, the functions return zero and num_chars_return is set to the number of rectangles required. Otherwise, the routines return a non-zero value.

If the overall_ink_return or overall_logical_return argument is non-NULL, XmbTextPerCharExtents and XwcTextPerCharExtents return the maximum extent of the string’s metrics to overall_ink_return or overall_logical_return, as returned by XmbTextExtents or XwcTextExtents.
13.5. Drawing Text Using Font Sets

The routines defined in this section draw text at a specified location in a drawable. They are similar to the functions `XDrawText`, `XDrawString`, and `XDrawImageString` except that they work with font sets instead of single fonts, and interpret the text based on the locale of the font set instead of treating the bytes of the string as direct font indexes. See section 8.6 for details of the use of GCs and possible protocol errors. If a `BadFont` error is generated, characters prior to the offending character may have been drawn.

The text is drawn using the fonts loaded for the specified font set; the font in the GC is ignored, and may be modified by the routines. No validation that all fonts conform to some width rule is performed.

The text functions `XmbDrawText` and `XwcDrawText` use the following structures.

```c
typedef struct {
    char *chars;        /* pointer to string */
    int nchars;         /* number of characters */
    int delta;          /* pixel delta between strings */
    XFontSet font_set;  /* fonts. None means don't change */
} XmbTextltem;

typedef struct {
    wchar_t *chars;      /* pointer to wide char string */
    int nchars;          /* number of wide characters */
    int delta;           /* pixel delta between strings */
    XFontSet font_set;   /* fonts. None means don't change */
} XwcTextltem;
```

To draw text using multiple font sets in a given drawable, use `XmbDrawText` or `XwcDrawText`.

```c
void XmbDrawText(Display *display, Drawable d, GC gc, XmbTextltem *items, int nitems);
void XwcDrawText(Display *display, Drawable d, GC gc, XwcTextltem *items, int nitems);
```

**display** Specifies the connection to the X server.

**d** Specifies the drawable.

**gc** Specifies the GC.

**x** Specify the x and y coordinates.

**y** Specify the x and y coordinates.

**items** Specifies an array of text items.

**nitems** Specifies the number of text items in the array.

`XmbDrawText` and `XwcDrawText` allow complex spacing and font set shifts between text strings. Each text item is processed in turn, with the origin of a text element advanced in the
primary draw direction by the escapement of the previous text item. A text item delta specifies
an additional escapement of the text item drawing origin in the primary draw direction. A
font_set member other than None in an item causes the font set to be used for this and subse-
quently text items in the text_items list. Leading text items with font_set member set to None
will not be drawn.

XmbDrawText and XwcDrawText do not perform any context-dependent rendering between
text segments. Clients may compute the drawing metrics by passing each text segment to
XmbTextExtents and XwcTextExtents or XmbTextPerCharExtents and XwcTextPerCharExtents. When the XFontSet has missing charsets, each unavailable character is drawn
with the default string returned by XCreateFontSet. The behavior for an invalid codepoint is
undefined.

To draw text using a single font set in a given drawable, use XmbDrawString or XwcDraw-
String.

```c
void XmbDrawString(Display *display, Drawable d, XFontSet font_set, GC gc, int x, int y, char *string, int num_bytes)

void XwcDrawString(Display *display, Drawable d, XFontSet font_set, GC gc, int x, int y, wchar_t *string, int num_wchars)
```

display
Specifies the connection to the X server.

d
Specifies the drawable.

font_set
Specifies the font set.

gc
Specifies the GC.

x
Specify the x and y coordinates.

y

string
Specifies the character string.

num_bytes
Specifies the number of bytes in the string argument.

num_wchars
Specifies the number of characters in the string argument.

XmbDrawString and XwcDrawString draw the specified text with the foreground pixel.
When the XFontSet has missing charsets, each unavailable character is drawn with the default
string returned by XCreateFontSet. The behavior for an invalid codepoint is undefined.

To draw image text using a single font set in a given drawable, use XmbDrawImageString or
XwcDrawImageString.
void XmbDrawImageString(dispplay, d, font_set, gc, x, y, string, num_bytes)
Display *display;
Drawable d;
XFontSet font_set;
GC gc;
int x, y;
char *string;
int num_bytes;

void XwcDrawImageString(display, d, font_set, gc, x, y, string, num_wchars)
Display *display;
Drawable d;
XFontSet font_set;
GC gc;
int x, y;
wchar_t *string;
int num_wchars;

display Specifies the connection to the X server.
d Specifies the drawable.
font_set Specifies the font set.
gc Specifies the GC.
x Specify the x and y coordinates.
y string Specifies the character string.
um_bytes Specifies the number of bytes in the string argument.
um_wchars Specifies the number of characters in the string argument.

XmbDrawImageString and XwcDrawImageString fill a destination rectangle with the background pixel defined in the GC and then paint the text with the foreground pixel. The filled rectangle is the rectangle returned to overall_logical_return by XmbTextExtents or XwcTextExtents for the same text and XFontSet.

When the XFontSet has missing charsets, each unavailable character is drawn with the default string returned by XCreateFontSet. The behavior for an invalid codepoint is undefined.

13.6. Input Method Overview
This section provides definitions for terms and concepts used for internationalized text input and a brief overview of the intended use of the mechanisms provided by Xlib.

A large number of languages in the world use alphabets consisting of a small set of symbols (letters) to form words. To enter text into a computer in an alphabetic language, a user usually has a keyboard on which there exists key symbols corresponding to the alphabet. Sometimes, a few characters of an alphabetic language are missing on the keyboard. Many computer users, who speak a Latin alphabet based language only have an English-based keyboard. They need to hit a combination of keystrokes in order to enter a character that does not exist directly on the keyboard. A number of algorithms have been developed for entering such characters, known as European input methods, the compose input method, or the dead-keys input method.

Japanese is an example of a language with a phonetic symbol set, where each symbol represents a specific sound. There are two phonetic symbol sets in Japanese: Katakana and Hiragana. In general, Katakana is used for words that are of foreign origin, and hiragana for writing native Japanese words. Collectively, the two systems are called Kana. Each set consists of 48 characters.
Korean also has a phonetic symbol set, called Hangul. Each of the 24 basic phonetic symbols (14 consonants and 10 vowels) represents a specific sound. A syllable is composed of two or three parts: the initial consonants, the vowels, and the optional last consonants. With Hangul, syllables can be treated as the basic units on which text processing is done. For example, a delete operation may work on a phonetic symbol or a syllable. Korean code sets include several thousands of these syllables. A user types the phonetic symbols that make up the syllables of the words to be entered. The display may change as each phonetic symbol is entered. For example, when the second phonetic symbol of a syllable is entered, the first phonetic symbol may change its shape and size. Likewise, when the third phonetic symbol is entered, the first two phonetic symbols may change their shape and size.

Not all languages rely solely on alphabetic or phonetic systems. Some languages, including Japanese and Korean, employ an ideographic writing system. In an ideographic system, rather than taking a small set of symbols and combining them in different ways to create words, each word consists of one unique symbol (or, occasionally, several symbols). The number of symbols may be very large: approximately 50,000 have been identified in Hanzi, the Chinese ideographic system.

There are two major aspects of ideographic systems for their computer usage. First, the standard computer character sets in Japan, China, and Korea include roughly 8,000 characters, while sets in Taiwan have between 15,000 and 30,000 characters, which make it necessary to use more than one byte to represent a character. Second, it obviously is impractical to have a keyboard that includes all of a given language's ideographic symbols. Therefore a mechanism is required for entering characters so that a keyboard with a reasonable number of keys can be used. Those input methods are usually based on phonetics, but there also exist methods based on the graphical properties of characters.

In Japan, both Kana and Kanji are used. In Korea, Hangul and sometimes Hanja are used. Now consider entering ideographs in Japan, Korea, China, and Taiwan.

In Japan, either Kana or English characters are typed and then a region is selected (sometimes automatically) for conversion to Kanji. Several Kanji characters may have the same phonetic representation. If that is the case with the string entered, a menu of characters is presented and the user must choose the appropriate one. If no choice is necessary or a preference has been established, the input method does the substitution directly. When Latin characters are converted to Kana or Kanji, it is called a romaji conversion.

In Korea, it is usually acceptable to keep Korean text in Hangul form, but some people may choose to write Hanja-originated words in Hanja rather than in Hangul. To change Hangul to Hanja, a region is selected for conversion and then the same basic method as described for Japanese is followed.

Probably because there are well-accepted phonetic writing systems for Japanese and Korean, computer input methods in these countries for entering ideographs are fairly standard. Keyboard keys have both English characters and phonetic symbols engraved on them, and the user can switch between the two sets.

The situation is different for Chinese. While there is a phonetic system called Pinyin promoted by authorities, there is no consensus for entering Chinese text. Some vendors use a phonetic decomposition (Pinyin or another), others use ideographic decomposition of Chinese words, with various implementations and keyboard layouts. There are about 16 known methods, none of which is a clear standard.

Also, there are actually two ideographic sets used: Traditional Chinese (the original written Chinese) and Simplified Chinese. Several years ago, the People's Republic Of China launched a campaign to simplify some ideographic characters and eliminate redundancies altogether. Under the plan, characters would be streamlined every five years. Characters have been revised several times now, resulting in the smaller, simpler set that makes up Simplified Chinese.
13.6.1. Input Method Architecture

As shown in the previous section, there are many different input methods in use today, each varying with language, culture, and history. A common feature of many input methods is that the user may type multiple keystrokes in order to compose a single character (or set of characters). The process of composing characters from keystrokes is called *preediting*. It may require complex algorithms and large dictionaries involving substantial computer resources. Input methods may require one or more areas in which to show the feedback of the actual keystrokes, to propose disambiguation to the user, to list dictionaries, and so on. The input method areas of concern are as follows.

- The Status area is intended to be a logical extension of the LEDs that exist on the physical keyboard. It is a window which is intended to present the internal state of the input method that is critical to the user. The status area may consist of text data and bitmaps or some combination.
- The Preedit area is intended to display the intermediate text for those languages that are composing prior to the client handling the data.
- The Auxiliary area is used for pop-up menus and customizing dialogs that may be required for an input method. There may be multiple Auxiliary areas for any input method. Auxiliary areas are managed by the input method independent of the client. Auxiliary areas are assumed to be a separate dialog which is maintained by the input method.

There are various user interaction styles used for preediting. The ones supported by Xlib are as follows.

- For on-the-spot input methods, preediting data will be displayed directly in the application window. Application data is moved to allow preedit data to be displayed at the point of insertion.
- Over-the-spot preediting means that the data is displayed in a preedit window that is placed over the point of insertion.
- Off-the-spot preediting means that the preedit window is inside the application window but not at the point of insertion. Often, this type of window is placed at the bottom of the application window.
- Root-window preediting refers to input methods that use a preedit window that is the child of RootWindow.

It would require a lot of computing resources if portable applications had to include input methods for all the languages in the world. To avoid this, a goal of the Xlib design is to allow an application to communicate with an input method placed in a separate process. Such a process is called an input server. The server to which the application should connect is dependent on the environment when the application is started up: what is the user language, the actual encoding to be used for it. The input method connection is said to be locale dependent. It is also user dependent: for a given language, the user can choose to some extent the user interface style of input method (if choice is possible among several).

Using an input server implies communication overhead, but applications can be migrated without relinking. Input methods can be implemented either as a stub communicating to an input server or as a local library.

An input method may be based on a front-end or a back-end architecture. In front-end, there are two separate connections to the X server: keystrokes go directly from X server to the input method on one connection, other events to the regular client connection. The input method is then acting as a filter, and sends composed strings to the client. Front-end requires synchronization between the two connections to avoid lost key events or locking issues.

In back-end, a single X server connection is used. A dispatching mechanism must decide on this channel to delegate appropriate keystrokes to the input method. For instance, it may retain
a Help keystroke for its own purpose. In the case where the input method is a separate process (that is, a server), there must be a special communication protocol between the back-end client and the input server.

Front-end introduces synchronization issues and filtering mechanism for non-character keystrokes (Functions, Help, and so on). Back-end sometimes implies more communication overhead and more process switching. If all three processes are running on a single workstation (X server, input server, client), there are two process switches for each keystroke in back-end, but there is only one in front-end.

The abstraction used by a client to communicate with an input method is an opaque data structure represented by the XIM data type. This data structure is returned by the XOpenIM function, which opens an input method on a given display. Subsequent operations on this data structure encapsulate all communication between client and input method. There is no need for an X client to use any networking library or natural language package in order to use an input method.

A single input server may be used for one or more languages, supporting one or more encoding schemes. But the strings returned from an input method will always be encoded in the (single) locale associated with XIM object.

### 13.6.2. Input Contexts

Xlib provides the ability to manage a multithreaded state for text input. A client may be using multiple windows, each window with multiple text entry areas, and the user possibly switching among them at any time. The abstraction for representing state of a particular input thread is called an input context. The Xlib representation of an input context is an XIC.

An input context is the abstraction retaining the state, properties, and semantics of communication between a client and an input method. An input context is a combination of an input method, a locale specifying the encoding of the character strings to be returned, a client window, internal state information, and various layout or appearance characteristics. The input context concept somewhat matches for input the graphics context abstraction defined for graphics output.

One input context belongs to exactly one input method. Different input contexts may be associated with the same input method, possibly with the same client window. An XIC is created with the XCreateIC function, providing an XIM argument, affiliating the input context to the input method for its lifetime. When an input method is closed with XCloseIM, all of its affiliated input contexts should not be used any more (and should preferably be destroyed before closing the input method).

Considering the example of a client window with multiple text entry areas, the application programmer could for example choose to implement:

- As many input contexts are created as text entry areas and the client will get the input accumulated on each context each time it will lookup that context.
- A single context is created for a top level window in the application. If such window contains several text entry areas, each time the user moves to another text entry area, the client has to indicate changes in the context.

A range of choices can be made by application designers to use either a single or multiple input contexts, according to the needs of their application.

### 13.6.3. Getting Keyboard Input

In order to obtain characters from an input method a client must call the function XmbLookupString or XwcLookupString with an input context created from that input method. Both a locale and display are bound to an input method when it is opened, and an input context inherits this locale and display. Any strings returned by XmbLookupString or XwcLookupString will be encoded in that locale.
13.6.4. Focus Management

For each text entry area in which the XmbLookupString or XwcLookupString routines are used there will be an associated input context.

When the application focus moves to a text entry area, the application must set the input context focus to the input context associated with that area. The input context focus is set by calling XSetICFocus with the appropriate input context.

Also, when the application focus moves out of a text entry area, the application should unset the focus for the associated input context by calling XUnsetICFocus. As an optimization, if XSetICFocus is called successively on two different input contexts, setting the focus on the second will automatically unset the focus on the first.

Note that in order to set and unset the input context focus correctly, it will be necessary to track application-level focus changes. Such focus changes do not necessarily correspond to X server focus changes.

If a single input context is being used to do input for multiple text entry areas, it will also be necessary to set the focus window of the input context whenever the focus window changes (see XNFocusWindow under XSetICValues).

13.6.5. Geometry Management

In most input method architectures (on-the-spot being the notable exception), the input method will perform the display of its own data. In order to provide better visual locality, it is often desirable to have the input method areas embedded within a client. In order to do this the client may need to allocate space for an input method. Xlib provides support that allows the size and position of input method areas to be provided by a client. The input method areas that are supported for geometry management are the Status area and the Preedit area.

The fundamental concept on which geometry management for input method windows is based is the proper division of responsibilities between the client (or toolkit) and the input method. The division of responsibilities is as follows:

- The client is responsible for the geometry of the input method window.
- The input method is responsible for the contents of the input method window.

An input method is able to suggest a size to the client, but it cannot suggest a placement. Also the input method can only suggest a size. It does not determine the size, and it must accept the size it is given.

Before a client provides geometry management for an input method, it must determine if geometry management is needed. The input method indicates the need for geometry management by setting XIMPreeditArea or XIMStatusArea in its XIMStyles value returned by XGetIMValues. When a client has decided that it will provide geometry management for an input method, it indicates that decision by setting the XNInputStyle value in the XIC.

After a client has established with the input method that it will do geometry management, the client must negotiate the geometry with the input method. The geometry is negotiated by the following steps:

- The client suggests an area to the input method by setting the XNAreaNeeded value for that area. If the client has no constraints for the input method it either will not suggest an area or will set the width and height to zero. Otherwise it will set one of the values.
- The client will get the XIC value XNAreaNeeded. The input method will return its suggested size in this value. The input method should pay attention to any constraints suggested by the client.
- The client sets the XIC value XNArea to inform the input method of the geometry of its window. The client should try to honor the geometry requested by the input method. The input method must accept this geometry.
Clients doing geometry management must be aware that setting other IC values may affect the geometry desired by an input method. For example, XNFontSet and XNLineSpacing may change the geometry desired by the input method.

The table of XIC values (see section 13.10) indicates the values that can cause the desired geometry to change when they are set. It is the responsibility of the client to renegotiate the geometry of the input method window when it is needed.

In addition, a geometry management callback is provided by which an input method can initiate a geometry change.

### 13.6.6. Event Filtering

A filtering mechanism is provided to allow input methods to capture X events transparently to clients. It is expected that toolkits (or clients) using `XmbLookupString` or `XwcLookupString` will call this filter at some point in the event processing mechanism to make sure that events needed by an input method can be filtered by that input method.

If there were no filter, a client could receive and discard events that are necessary for the proper functioning of an input method. The following provides a few examples of such events:

- Expose events on preedit window in local mode.
- Events may be used by an input method to communicate with an input server. Such input server protocol related events have to be intercepted if one does not want to disturb client code.
- Key events can be sent to a filter before they are bound to translations such as Xt provides.

Clients are expected to get the XIC value `XNFilterEvents` and augment the event mask for the client window with that event mask. This mask may be zero.

### 13.6.7. Callbacks

When an on-the-spot input method is implemented, only the client can insert or delete preedit data in place and possibly scroll existing text. This means the echo of the keystrokes has to be achieved by the client itself, tightly coupled with the input method logic.

When a keystroke is entered, the client calls `XmbLookupString` or `XwcLookupString`. At this point, in the on-the-spot case, the echo of the keystroke in the preedit has not yet been done. Before returning to the client logic that handles the input characters, the lookup function must call the echoing logic for inserting the new keystroke. If the keystrokes entered so far make up a character, the keystrokes entered need to be deleted, and the composed character will be returned. Hence, what happens is that, while being called by client code, input method logic has to call back to the client before it returns. The client code, that is, a callback routine, is called from the input method logic.

There are a number of cases where the input method logic has to call back the client. Each of those cases is associated with a well-defined callback action. It is possible for the client to specify, for each input context, what callback is to be called for each action.

There are also callbacks provided for feedback of status information and a callback to initiate a geometry request for an input method.

### 13.7. Variable Argument Lists

Several input method functions have arguments which conform to ANSI C variable argument list calling convention. Each function denoted with a "..." argument takes a variable length list of name and value pairs where each name is a string and each value is of type `XPointer`. The end of the list is identified by a name argument containing NULL.
A variable length argument list may contain a nested list. If the name XVaNestedList is specified in place of an argument name, then the following value is interpreted as a XVaNestedList value which specifies a list of values logically inserted into the original list at the point of declaration. The end of a nested list is identified with a NULL.

To allocate a nested variable argument list dynamically, use XVaCreateNestedList.

```c
typedef void * XVaNestedList;
XVaNestedList XVaCreateNestedList(dummy, ...)
    int dummy;
    Unused argument (required by ANSI C).
    ...  Specifies the variable length argument list.
```

The XVaCreateNestedList function allocates memory and copies its arguments into a single list pointer which may be used as value for arguments requiring a list value. Any entries are copied as specified. Data passed by reference is not copied; the caller must ensure data remains valid for the lifetime of the nested list. The list should be freed using XFree when it is no longer needed.

### 13.8. Input Method Functions

To open a connection, use XOpenIM.

```c
XIM XOpenIM(display, db, res_name, res_class)
    Display *display;
    XrmDataBase db;
    char *res_name;
    char *res_class;
    display  Specifies the connection to the X server.
    db       Specifies a pointer to the resource database.
    res_name Specifies the full resource name of the application.
    res_class Specifies the full class name of the application.
```

The XOpenIM function opens an input method, matching the current locale and modifiers specification. Current locale and modifiers are bound to the input method at opening time. The locale associated with an input method cannot be changed dynamically. This implies the strings returned by XmbLookupString or XwcLookupString, for any input context affiliated with a given input method, will be encoded in the locale current at the time input method is opened.

The specific input method to which this call will be routed is identified on the basis of the current locale. XOpenIM will identify a default input method corresponding to the current locale. That default can be modified using XSetLocaleModifiers for the input method modifier.

The db argument is the resource database to be used by the input method for looking up resources that are private to the input method. It is not intended that this database be used to look up values that can be set as IC values in an input context. If db is NULL, no data base is passed to the input method.

The res_name and res_class arguments specify the resource name and class of the application. They are intended to be used as prefixes by the input method when looking up resources that are common to all input contexts that may be created for this input method. The characters used for resource names and classes must be in the X portable character set. The resources looked up are not fully specified if res_name or res_class is NULL.
The res_name and res_class arguments are not assumed to exist beyond the call to XOpenIM. The specified resource database is assumed to exist for the lifetime of the input method. XOpenIM returns NULL if no input method could be opened.

To close a connection, use XCloseIM.

Status XCloseIM(im)

im Specifies the input method.

The XCloseIM function closes the specified input method.

To query an input method, use XGetIMValues.

char * XGetIMValues(im, ...)

im Specifies the input method.
...
Specifies the variable length argument list to get XIM values.

The XGetIMValues function presents a variable argument list programming interface for querying properties or features of the specified input method. This function returns NULL if it succeeds; otherwise, it returns the name of the first argument that could not be obtained.

Only one standard argument is defined by Xlib: XNQueryInputStyle, which must be used to query about input styles supported by the input method.

A client should always query the input method to determine which styles are supported. The client should then find an input style it is capable of supporting.

If the client cannot find an input style that it can support it should negotiate with the user the continuation of the program (exit, choose another input method, and so on).

The argument value must be a pointer to a location where the returned value will be stored. The returned value is a pointer to a structure of type XIMStyles. Clients are responsible for freeing the XIMStyles data structure. To do so, use XFree.

The XIMStyles structure is defined as follows.

typedef struct {
    unsigned short count_styles;
    XIMStyle * supported_styles;
} XIMStyles;

An XIMStyles structure contains in its field count_styles, the number of input styles supported. This is also the size of the array in the field supported_styles.

The supported styles is a list of bit mask combinations, which indicate the combination of styles for each of the areas supported. These areas are described below. Each element in the
list should select one of the bit mask values for each area. The list describes the complete set
of combinations supported. Only these combinations are supported by the input method.

The Preedit category defines what type of support is provided by the input method for preedit information:

- **XIMPreeditArea** If chosen, the input method would require the client to provide some area values for it to do its preediting. Refer to XIC values XNArea and XNAreaNeeded.
- **XIMPreeditPosition** If chosen, the input method would require the client to provide positional values. Refer to XIC values XNSpotLocation and XNFocusWindow.
- **XIMPreeditCallbacks** If chosen, the input method would require the client to define the set of preedit callbacks. Refer to XIC values XNPreeditStartCallback, XNPreeditDoneCallback, XNPreeditDrawCallback, and XNPreeditCaretCallback.
- **XIMPreeditNothing** If chosen, the input method can function without any Preedit values.
- **XIMPreeditNone** The input method does not provide any Preedit feedback. Any Preedit value is ignored. This style is mutually exclusive with the other Preedit styles.

The Status category defines what type of support is provided by the input method for status information:

- **XIMStatusArea** The input method requires the client to provide some area values for it to do its Status feedback. See XNArea and XNAreaNeeded.
- **XIMStatusCallbacks** The input method requires the client to define the set of status callbacks; XNStatusStartCallback, XNStatusDoneCallback, and XNStatusDrawCallback.
- **XIMStatusNothing** The input method can function without any Status values.
- **XIMStatusNone** The input method does not provide any Status feedback. If chosen, any Status value is ignored. This style is mutually exclusive with the other Status styles.

To obtain the display associated with an input method, use XDisplayOfIM.

```c
Display * XDisplayOfIM(im)
    XIM im;
```

`im` Specifies the input method.

The `XDisplayOfIM` function returns the display associated with the specified input method.

To get the locale associated with an input method, use XLocaleOfIM.

```c
char * XLocaleOfIM(im)
    XIM im;
```

`im` Specifies the input method.

The `XLocaleOfIM` returns the locale associated with the specified input method.

### 13.9. Input Context Functions

An input context is an abstraction that is used to contain both the data required (if any) by an input method and the information required to display that data. There may be multiple input contexts for one input method. The programming interfaces for creating, reading, or modifying an input context use a variable argument list. The name elements of the argument lists are
referred to as XIC values. It is intended that input methods be controlled by these XIC values. As new XIC values are created they should be registered with the X Consortium.

To create an input context use XCreateIC.

```
XIC XCreateIC(im, ...)  
    XIM im;
```

- `im` Specifies the input method.
- `...` Specifies the variable length argument list to set XIC values.

The XCreateIC function creates a context within the specified input method. Some of the arguments are mandatory at creation time, and the input context will not be created if they are not provided. Those arguments are the input style and the set of text callbacks (if the input style selected requires callbacks). All other input context values can be set later.

XCreateIC returns a NULL value if no input context could be created. A NULL value could be returned for any of the following reasons:
- A required argument was not set.
- A read-only argument was set (for example, XNFilterEvents).
- The argument name is not recognized.
- The input method encountered an input method implementation dependent error.

XCreateIC can generate BadAtom, BadColor, BadPixmap, and BadWindow errors.

To destroy an input context, use XDestroyIC.

```
void XDestroyIC(ic)    
    XIC ic;
```

- `ic` Specifies the input context.

XDestroyIC destroys the specified input context.

To communicate to and synchronize with input method for any changes in keyboard focus from the client side, use XSetICFocus and XUnsetICFocus.

```
void XSetICFocus(ic)    
    XIC ic;
```

- `ic` Specifies the input context.

The XSetICFocus function allows a client to notify an input method that the focus window attached to the specified input context has received keyboard focus. The input method should take action to provide appropriate feedback. Complete feedback specification is a matter of user interface policy.

```
void XUnsetICFocus(ic)    
    XIC ic;
```

- `ic` Specifies the input context.

The XUnsetICFocus function allows a client to notify an input method that the specified input context has lost the keyboard focus and that no more input is expected on the focus window attached to that input context. The input method should take action to provide appropriate feedback. Complete feedback specification is a matter of user interface policy.

To reset the state of an input context to initial state, use XmbResetIC or XwcResetIC.
char * XmbResetIC(ic)
XIC ic;

wchar_t * XwcResetIC(ic)
XIC ic;

ic Specifies the input context.
The XmbResetIC and XwcResetIC functions reset input context to initial state. Any input pending on that context is deleted. Input method is required to clear preedit area, if any, and update status accordingly. Calling XmbResetIC or XwcResetIC does not change the focus. The return value of XmbResetIC is its current preedit string as a multibyte string. The return value of XwcResetIC is its current preedit string as a wide character string. It is input method implementation dependent whether these routines return a non-NULL string or NULL. The client should free the returned string by calling XFree.

To get the input method associated with an input context, use XIMOOfIC.
XIM XIMOOfIC(ic)
XIC ic;

ic Specifies the input context.
The XIMOOfIC function returns the input method associated with the specified input context.

Xlib provides two functions for setting and reading XIC values, respectively: XSetICValues and XGetICValues. Both functions have a variable length argument list. In that argument list, any XIC value’s name must be denoted with a character string using the X Portable Character Set.

To set XIC values, use XSetICValues.
char * XSetICValues(ic, ...)
XIC ic;

ic Specifies the input context.
... Specifies the variable length argument list to set XIC values.
The XSetICValues function returns NULL if no error occurred; otherwise, it returns the name of the first argument that could not be set. An argument could be not set for any of the following reasons:
• A read-only argument was set (for example, XNFilterEvents).
• The argument name is not recognized.
• The input method encountered an input method implementation dependent error.
Each value to be set must be an appropriate datum, matching the data type imposed by the semantics of the argument. XSetICValues can generate BadAtom, BadColor, BadCursor, BadPixmap, and BadWindow errors.

To obtain XIC values, use XGetICValues.
char * XGetICValues(ic, ...)
XIC ic;

ic Specifies the input context.
... Specifies the variable length argument list to get XIC values.
The XGetICValues function returns NULL if no error occurred; otherwise, it returns the name of the first argument that could not be obtained. An argument could be not obtained for any of the following reasons:

- The argument name is not recognized.
- The input method encountered an implementation dependent error.

Each argument value (following a name) must point to a location where the value is to be stored. XGetICValues allocates memory to store the values, and client is responsible for freeing each value by calling XFree.

### 13.10. XIC Value Arguments

The following tables describe how XIC values are interpreted by an input method depending on the input style chosen by the user.

The first column lists the XIC values. The second column indicates which values are involved in affecting, negotiating, and setting the geometry of the input method windows. The subentries under the third column indicate the different input styles that are supported. Each of these columns indicates how each of the XIC values are treated by that input style.

The following Keys apply to these tables.

<table>
<thead>
<tr>
<th>Keys</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>This value must be set with XCreateIC.</td>
</tr>
<tr>
<td>D</td>
<td>This value may be set using XCreateIC. If it is not set, a default is provided.</td>
</tr>
<tr>
<td>G</td>
<td>This value may be read using XGetICValues.</td>
</tr>
<tr>
<td>GN</td>
<td>This value may cause geometry negotiation when its value is set by means of XCreateIC or XSetICValues.</td>
</tr>
<tr>
<td>GR</td>
<td>This value will be the response of the input method when any GN value is changed.</td>
</tr>
<tr>
<td>GS</td>
<td>This value will cause the geometry of the input method window to be set.</td>
</tr>
<tr>
<td>O</td>
<td>This value must be set once and only once. It need not be set at create time.</td>
</tr>
<tr>
<td>S</td>
<td>This value may be set with XSetICValues.</td>
</tr>
<tr>
<td>ignored</td>
<td>This value is ignored by the input method for the given input style.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XIC Value</th>
<th>Geometry Management</th>
<th>Preedit Callback</th>
<th>Input Style</th>
<th>Preedit Position</th>
<th>Preedit Area</th>
<th>Preedit Nothing</th>
<th>Preedit None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Style</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client Window</td>
<td>O-G</td>
<td>C-G</td>
<td>C-G</td>
<td>C-G</td>
<td>C-G</td>
<td>C-G</td>
<td>C-G</td>
</tr>
<tr>
<td>Focus Window</td>
<td>GN</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>ignored</td>
</tr>
<tr>
<td>Resource Name</td>
<td>ignored</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>ignored</td>
<td>ignored</td>
</tr>
<tr>
<td>Resource Class</td>
<td>ignored</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>ignored</td>
<td>ignored</td>
</tr>
<tr>
<td>Geometry Callback</td>
<td>ignored</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>ignored</td>
<td>ignored</td>
</tr>
<tr>
<td>FilterEvents</td>
<td>G</td>
<td>ignored</td>
<td>ignored</td>
<td>D-S-G</td>
<td>ignored</td>
<td>ignored</td>
<td>ignored</td>
</tr>
<tr>
<td>Preedit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>GS</td>
<td>ignored</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>ignored</td>
<td>ignored</td>
<td>ignored</td>
</tr>
<tr>
<td>AreaNeeded</td>
<td>GN-GR</td>
<td>ignored</td>
<td>ignored</td>
<td>S-G</td>
<td>ignored</td>
<td>ignored</td>
<td>ignored</td>
</tr>
<tr>
<td>SpooLocation</td>
<td>ignored</td>
<td>C-S-G</td>
<td>ignored</td>
<td>ignored</td>
<td>ignored</td>
<td>ignored</td>
<td>ignored</td>
</tr>
<tr>
<td>Colormap</td>
<td>ignored</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>ignored</td>
<td>ignored</td>
</tr>
<tr>
<td>Foreground</td>
<td>ignored</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>ignored</td>
<td>ignored</td>
</tr>
<tr>
<td>Background</td>
<td>ignored</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>D-S-G</td>
<td>ignored</td>
<td>ignored</td>
</tr>
</tbody>
</table>

246
13.10.1. Input Style

The XNInputStyle argument specifies the input style to be used. The value of this argument must be one of the values returned by the XGetIMValues function with the XNQueryInputStyle argument specified in the supported_styles list.

Note that this argument must be set at creation time and cannot be changed.

13.10.2. Client Window

The XNClientWindow argument specifies to the input method the client window in which the input method can display data or create subwindows. Geometry values for input method areas are given with respect to the client window. Dynamic change of client window is not supported. Note that this argument may be set only once and should be set before any input is done using this input context. If it is not set the input method may not operate correctly.

If an attempt is made to set this value a second time with XSetICValues, the string XNClientWindow will be returned by XSetICValues, and the client window will not be changed.
If the client window is not a valid window ID on the display attached to the input method, a BadWindow error can be generated when this value is used by the input method.

13.10.3. Focus Window
The XNFocusWindow argument specifies the focus window. The primary purpose of the XNFocusWindow is to identify the window that will receive the key event when input is composed. In addition, the input method may possibly affect the focus window as follows:

• Select events on it
• Send events to it
• Modify its properties
• Grab keyboard within that window

The value associated to the argument must be of type Window. If the focus window is not a valid window ID on the display attached to the input method, a BadWindow error can be generated when this value is used by the input method.

When this XIC value is left unspecified, the input method will default focus window to client window.

13.10.4. Resource Name and Class
The XNResourceName and XNResourceClass arguments are strings that specify the full name and class used by the client to obtain resources for the client window. These values should be used as prefixes for name and class when looking up resources that may vary according to the input context. If these values are not set, the resources will not be fully specified.

It is not intended that values which can be set as XIC values be set as resources.

13.10.5. Geometry Callback
The XNGeometryCallback argument is a structure of type XIMCallback (see section 13.10.7.10).

The XNGeometryCallback argument specifies the geometry callback which a client can set. This callback is not required for correct operation of either an input method or a client. It can be set for a client whose user interface policy permits an input method to request the dynamic change of that input methods window. An input method that does dynamic change will need to filter any events that it uses to initiate the change.

13.10.6. Filter Events
The XNFilterEvents argument returns the event mask that an input methods needs to have selected for. The client is expected to augment its own event mask for the client window with this one.

Note that this argument is read-only, is set by the input method at create time, and is never changed.

Note also that the type of this argument is "unsigned long". Setting this value will cause an error.

13.10.7. Preedit and Status Attributes
The XNPreeditAttributes and XNStatusAttributes arguments specify to input method the attributes to be used for the Preedit and Status areas, if any. Those attributes are passed to XSetICValues or XGetICValues as a nested variable length list. The names to be used in these lists are as described in the following sections.
13.10.7.1. Area
The value of the XNArea argument must be a pointer to a structure of type XRectangle. The interpretation of the XNArea argument is dependent on the input method style that has been set.

If the input method style is XIMPreeditPosition, XNArea specifies the clipping region within which preediting will take place. If the focus window has been set, the coordinates are assumed to be relative to the focus window. Otherwise, the coordinates are assumed to be relative to the client window. If neither has been set, the results are undefined. If XNArea is not specified, the input method will default the clipping region to the geometry of the XNFocusWindow. If the area specified is NULL or invalid, the results are undefined.

If the input style is XIMPreeditArea or XIMStatusArea, XNArea specifies the geometry provided by the client to the input method. The input method may use this area to display its data, either Predit or Status depending on the area designated. The input method may create a window as a child of the client window with dimensions that fit the XNArea. The coordinates are relative to the client window. If the client window has not been set yet, the input method should save these values and apply them when the client window is set. If XNArea is not specified, is set to NULL or is invalid, the results are undefined.

13.10.7.2. Area Needed
When set, the XNAreaNeeded argument specifies the geometry suggested by the client for this area (Preedit or Status). The value associated with the argument must be a pointer to a structure of type XRectangle. Note that the x, y values are not used and that non-zero values for width or height are the constraints that the client wishes the input method to respect.

When read, the XNAreaNeeded argument specifies the preferred geometry desired by the input method for the area.

This argument is only valid if the input style is XIMPreeditArea or XIMStatusArea. It is used for geometry negotiation between the client and the input method and has no other effect upon the input method (see section 13.6.5).

13.10.7.3. Spot Location
The XNSpotLocation argument specifies to the input method the coordinates of the "spot" to be used by an input method executing with XNInputStyle set to XIMPreeditPosition. When specified to any input method other than XIMPreeditPosition, this XIC value is ignored.

The x coordinate specifies the position where the next character would be inserted. The y coordinate is the position of the baseline used by current text line in the focus window. The x and y coordinates are relative to the focus window, if it has been set; otherwise, they are relative to the client window. If neither the focus window nor the client window has been set, the results are undefined.

Note that the value of the argument is a pointer to a structure of type XPoint.

13.10.7.4. Colormap
Two different arguments can be used to indicate what colormap the input method should use to allocate colors: a colormap ID, or a standard colormap name.

The XNColormap argument is used to specify a colormap ID. The argument value is of type Colormap. An invalid argument may generate a BadCoSor error when it is used by the input method.

The XNStdColormap argument is used to indicate the name of the standard colormap in which input method should to allocate colors. The argument value is an Atom that should be a valid atom for calling XGetRGBColormaps. An invalid argument may generate a BadAtom error when it is used by the input method.
If colormap is left unspecified, it is defaulted to client window colormap.

13.10.7.5. Foreground and Background
The XNForeground and XNBackground arguments specify the foreground and background pixel, respectively. The argument value is of type "unsigned long". It must be a valid pixel in the input method colormap.
If these values are left unspecified, the default is determined by the input method.

13.10.7.6. Background Pixmap
The XNBackgroundPixmap argument specifies a background pixmap to be used as the background of the window. The value must be of type Pixmap. An invalid argument may generate a BadPixmap error when it is used by the input method.
If this value is left unspecified, the default is determined by the input method.

13.10.7.7. FontSet
The XNFontSet argument specifies to the input method what fontset is to be used. The argument value is of type XFontSet.
If this value is left unspecified, the default is determined by the input method.

13.10.7.8. Line Spacing
The XNLineSpace argument specifies to the input method what line spacing is to be used in preedit window if more than one line is to be used. This argument is of type "int".
If this value is left unspecified, the default is determined by the input method.

13.10.7.9. Cursor
The XNCursor argument specifies to the input method what cursor is to be used in the specified window. This argument is of type Cursor.
An invalid argument may generate a BadCursor error when it is used by the input method. If this value is left unspecified, the default is determined by the input method.

13.10.7.10. Preedit and Status Callbacks
A client that wishes to support the input style XIMPreeditCallbacks must provide a set of preedit callbacks to the input method. The set of preedit callbacks are as follows:

XNPreeditStartCallback This is called when the input method starts preedit.
XNPreeditDoneCallback This is called when the input method stops preedit.
XNPreeditDrawCallback This is called when a number preedit keystrokes should be echoed.
XNPreeditCaretCallback This is called to move text insertion point within preedit string

A client that wishes to support the input style XIMStatusCallbacks must provide a set of status callbacks to the input method. The set of status callbacks are as follows:

XNStatusStartCallback This is called when the input method initializes status area.
XNStatusDoneCallback This is called when the input method no longer needs status area.
XNStatusDrawCallback This is called when updating the status area is required.

The value of any status or preedit argument is a pointer to a structure of type XIMCallback.

typedef void (*XIMProc);
typedef struct {
Each callback has some particular semantics and will carry the data that expresses the environment necessary to the client into a specific data structure. This paragraph only describes the arguments to be used to set the callback. For a complete description of the semantics, see section 13.11.

Note that setting any of these values while doing preedit may cause unexpected results.

13.11. Callback Semantics

Callbacks are functions defined by clients or text drawing packages that are to be called from the input method when selected events occur. Most clients will use a text editing package or a toolkit and, hence, will not need to define such callbacks. This section defines the callback semantics, when they are triggered, and what their arguments are. Note that this information is mostly useful for X toolkit implementors.

Callbacks are mostly provided so that clients (or text editing packages) can implement on-the-spot preediting in their own window. In that case, the input method needs to communicate and synchronize with the client. Input method needs to communicate changes in the preedit window when it is under control of the client. Those callbacks allow the client to initialize the preedit area, display a new preedit string, move the text insertion point inside preedit, terminate preedit, or update the status area.

All callback functions follow the generic prototype:

```c
void CallbackPrototype(ic, client_data, call_data)
    XIC ic;
    XPointer client_data;
    SomeType call_data;
```

- `ic`: Specifies the input context.
- `client_data`: Specifies the additional client data.
- `call_data`: Specifies data specific to the callback.

The `call_data` argument is a structure that expresses the arguments needed to achieve the semantics; that is, it is a specific data structure appropriate to the callback. In cases where no data is needed in the callback, this `call_data` argument is NULL. The `client_data` argument is a closure that has been initially specified by the client when specifying the callback and passed back. It may serve, for example, to inherit application context in the callback.

The following paragraphs describe the programming semantics and specific data structure associated with the different reasons.

13.11.1. Geometry Callback

The geometry callback is triggered by the input method to indicate that it wants the client to negotiate geometry. The generic prototype is as follows:

```c
void GeometryCallback(ic, client_data, call_data)
    XIC ic;
    XPointer client_data;
    XPointer call_data;
```

- `ic`: Specifies the input context.
- `client_data`: Specifies the additional client data.
- `call_data`: Not used for this callback, always passed as NULL.
Note that a GeometryCallback is called with a NULL call_data argument.

### 13.11.2. Preedit State Callbacks

When the input method turns input conversion on or off, a PreeditStartCallback or PreeditDoneCallback callback is triggered in order to let the toolkit do the setup or the cleanup for the preedit region.

```c
int PreeditStartCallback(ic, client_data, call_data)
    XIC ic;
    XPointer client_data;
    XPointer call_data;

ic    Specifies the input context.
client_data Specifies the additional client data.
call_data     Not used for this callback, always passed as NULL.
```

When preedit starts on the specified input context, the callback is called with a NULL call_data argument. PreeditStartCallback will return the maximum size of the preedit string. Note that a positive number indicates the maximum number of bytes allowed in the preedit string, and a value of -1 indicates there is no limit.

```c
void PreeditDoneCallback(ic, client_data, call_data)
    XIC ic;
    XPointer client_data;
    XPointer call_data;

ic    Specifies the input context.
client_data Specifies the additional client data.
call_data     Not used for this callback, always passed as NULL.
```

When preedit stops on the specified input context, the callback is called with a NULL call_data argument. The client can release the data allocated by PreeditStartCallback.

PreeditStartCallback should initialize appropriate data needed for displaying preedit information and for handling further PreeditDrawCallback calls. Once PreeditStartCallback is called, it will not be called again before PreeditDoneCallback has been called.

### 13.11.3. PreeditDraw Callback

This callback is triggered to draw and insert, delete or replace, preedit text in the preedit region. The preedit text may include unconverted input text such as Japanese kana, converted text such as Japanese Kanji characters, or characters of both kinds. That string is either a multibyte or wide character string, whose encoding matches the locale bound to the input context. The callback prototype is as follows:

```c
void PreeditDrawCallback(ic, client_data, call_data)
    XIC ic;
    XPointer client_data;
    XIMPreeditDrawCallbackStruct *call_data;

ic    Specifies the input context.
client_data Specifies the additional client data.
call_data     Specifies the preedit drawing information.
```

The callback is passed a XIMPreeditDrawCallbackStruct structure in the call_data argument. The text member of this structure contains the text to be drawn. After the string has been drawn, the caret should be moved to the specified location.

The XIMPreeditDrawCallbackStruct structure is defined as follows:
typedef struct _XIMPreeditDrawCallbackStruct {
    int caret;       /* Cursor offset within preedit string */
    int chg_first;   /* Starting change position */
    int chg_length;  /* Length of the change in character count */
    XIMText *text;
} XIMPreeditDrawCallbackStruct;

The client must keep updating a buffer of the preedit text, the callback arguments referring to
indexes in that buffer. The call_data fields have specific meanings according to the operation:

- To indicate text deletion, the call_data specifies a NULL text field. The text to be
deleted is then the current text in buffer from position chg_first (starting at zero) on a
(character) length of chg_length.

- When text is non-NULL, it indicates insertion or replacement of text in the buffer.
  A positive chg_length indicates that the characters starting from chg_first to
  ch_first+chg_length must be deleted and must be replaced by text, whose length is
  specified in the XIMText structure.
  A chg_length value of zero indicates that text must be inserted right at the position
  specified by chg_first. A value of zero for chg_first specifies the first character in the
  buffer.

- The caret member is an index in the the preedit text buffer that specifies the character
  after which the cursor should move after text has been drawn or deleted.

typedef struct _XIMText {
    unsigned short length;
    XIMFeedback *feedback;
    Bool encoding_is_wchar;
    union {
        char * multi_byte;
        wchar_t * wide_char;
    } string;
} XIMText;

The text string passed is actually a structure specifying:

- The length member is the text length in characters.
- The encoding_is_wchar member is a value that indicates if the text string is encoded in
  wide character or multibyte format. This value should be set by the client when it sets
  the callback.
- The string member is the text string.
- The feedback member indicates rendering type.

The feedback member express the types of rendering feedback the callback should apply when
drawing text. Rendering of the text to be drawn is specified either in generic ways (for example,
primary, secondary) or in specific ways (reverse, underline). When generic indications are
given, the client is free to choose the rendering style. It is necessary however that primary and
secondary are mapped to two distinct rendering styles.

The feedback member also specifies how the rendering of the text argument should be
achieved. If feedback is NULL, then rendering is assumed to be the same as rendering of
other characters in the text entry. Otherwise, it specifies an array defining the rendering of
each character of the string (hence the length of the array is length).

If an input method wishes to indicate that it is only updating the feedback of the preedit text
without changing the content of it, the XIMText structure should contain a NULL value for
the string field, the number of characters affected in the length field, and the feedback field
should point to an array of XIMFeedback.
Each element in the array is a bit mask represented by a value of type XIMFeedback. The valid masks names are as follows:

```c
typedef unsigned long XIMFeedback;
```

```c
#define XIMReverse 1
#define XIMUnderline (1L<<1)
#define XIMHighlight (1L<<2)
#define XIMPrimary (1L<<3)
#define XIMSecondary (1L<<4)
#define XIMTertiary (1L<<5)
```

### 13.11.4. PreeditCaretCallback

An input method may have its own "navigation keys" to allow the user to move the text insertion point in the preedit area (for example, to move backward or forward). Consequently, input method needs to indicate to the client that it should move the text insertion point. It then calls the PreeditCaretCallback.

```c
void PreeditCaretCallback(ic, client_data, call_data)
    XIC ic;
    XPointer client_data;
    XIMPreeditCaretCallbackStruct *call_data;
```

- `ic`: Specifies the input context.
- `client_data`: Specifies the additional client data.
- `call_data`: Specifies the preedit caret information.

The input method will trigger PreeditCaretCallback to move the text insertion point during preedit. The call_data argument contains a pointer to an XIMPreeditCaretCallbackStruct structure, which indicates where the caret should be moved. The callback must move the insertion point to its new location and return, in field position, the new offset value from initial position.

The XIMPreeditCaretCallbackStruct structure is defined as follows:

```c
typedef struct _XIMPreeditCaretCallbackStruct {
    int position; /* Caret offset within preedit string */
    XIMCaretDirection direction; /* Caret moves direction */
    XIMCaretStyle style; /* Feedback of the caret */
} XIMPreeditCaretCallbackStruct;
```

The XIMCaretStyle structure is defined as follows:

```c
typedef enum {
    XIMIsInvisible, /* Disable caret feedback */
    XIMIsPrimary, /* UI defined caret feedback */
    XIMIsSecondary, /* UI defined caret feedback */
} XIMCaretStyle;
```

The XIMCaretDirection structure is defined as follows:

```c
typedef enum {
    XIMForwardChar, XIMBackwardChar,
    XIMForwardWord, XIMBackwardWord,
    XIMCaretUp, XIMCaretDown,
    XIMNextLine, XIMPreviousLine,
    XIMLineStart, XIMLineEnd,
    XIMAbsolutePosition,
    XIMDontChange,
} XIMCaretDirection;
```
The meaning of these values are defined as follows:

- **XIMForwardChar**: Move caret forward one character position.
- **XIMBackwardChar**: Move caret backward one character position.
- **XIMForwardWord**: Move caret forward one word position.
- **XIMBackwardWord**: Move caret backward one word position.
- **XIMCaretUp**: Move caret up one line keeping current offset.
- **XIMCaretDown**: Move caret down one line keeping current offset.
- **XIMPreviousLine**: Move caret up one line.
- **XIMNextLine**: Move caret down one line.
- **XIMLineStart**: Move caret to the beginning of the current display line that contains the caret.
- **XIMLineEnd**: Move caret to the end of the current display line that contains the caret.
- **XIMAbsolutePosition**: The callback must move to the location specified by the position field of the callback data, indicated in characters, starting from the beginning of the preedit text. Hence, a value of zero means move back to beginning of the preedit text.
- **XIMDontChange**: The caret position does not change.

13.11.5. Status Callbacks

An input method may communicate changes in the status of an input context (for example, created, destroyed, or focus changes) with three status callbacks: StatusStartCallback, StatusDoneCallback, and StatusDrawCallback.

When the input context is created or gains focus, the input method calls the StatusStartCallback callback.

```c
void StatusStartCallback(ic, client_data, call_data)
    XIC ic;
    XPointer client_data;
    XPointer call_data;
```

- `ic` Specifies the input context.
- `client_data` Specifies the additional client data.
- `call_data` Not used for this callback, always passed as NULL.

The callback should initialize appropriate data for displaying status and be prepared to further StatusDrawCallback calls. Once StatusStartCallback is called, it will not be called again before StatusDoneCallback has been called.

When an input context is destroyed or when it loses focus, the input method calls StatusDoneCallback.

```c
void StatusDoneCallback(ic, client_data, call_data)
    XIC ic;
    XPointer client_data;
    XPointer call_data;
```

- `ic` Specifies the input context.
- `client_data` Specifies the additional client data.
- `call_data` Not used for this callback, always passed as NULL.
The callback may release any data allocated on StatusStart.

When an input context status has to be updated, the input method calls StatusDrawCallback.

```c
void StatusDrawCallback(ic, client_data, call_data)
    XIC ic;
    XPointer client_data;
    XIMStatusDrawCallbackStruct *call_data;
```

- `ic` Specifies the input context.
- `client_data` Specifies the additional client data.
- `call_data` Specifies the status drawing information.

The callback should update the status area by either drawing a string, or imaging a bitmap in the status area.

The XIMStatusDataType and XIMStatusDrawCallbackStruct structures are defined as follows:

```c
typedef enum {
    XIMTextType,
    XIMBitmapType,
} XIMStatusDataType;

typedef struct _XIMStatusDrawCallbackStruct {
    XIMStatusDataType type;
    union {
        XIMText *text;
        Pixmap bitmap;
    } data;
} XIMStatusDrawCallbackStruct;
```

### 13.12. Event Filtering

Xlib provides the ability for an input method to register a filter internal to Xlib. This filter is called by a client (or toolkit) by calling XFilterEvent after calling XNextEvent. Any client that uses the XIM interface should call XFilterEvent to allow input methods to process their events without knowledge of the client’s dispatching mechanism. A client’s user interface policy may determine the priority of event filters with respect to other event handling mechanisms (for example, modal grabs).

Clients may not know how many filters there are, if any, and what they do. They may only know if an event has been filtered on return of XFilterEvent. Clients should discard filtered events.

```c
Bool XFilterEvent(event, w)
    XEvent *event;
    Window w;
```

- `event` Specifies the event to filter.
- `w` Specifies the window for which the filter is to be applied.

If the window argument is None, XFilterEvent applies the filter to the window specified in the XEvent structure. The window argument is provided so that layers above Xlib that do event redirection can indicate to which window an event has been redirected.

If XFilterEvent returns True, then some input method has filtered the event, and the client should discard the event. If XFilterEvent returns False, then the client should continue processing the event.
If a grab has occurred in the client, and XFilterEvent returns True, the client should ungrab the keyboard.

13.13. Getting Keyboard Input

To get composed input from an input method, use XmbLookupString or XwcLookupString.

```c
int XmbLookupString(ic, event, buffer_return, bytes_buffer, keysym_return, status_return)
    XIC ic;
    XKeyPressEvent *event;
    char *buffer_return;
    int bytes_buffer;
    KeySym *keysym_return;
    Status *status_return;
```

```c
int XwcLookupString(ic, event, buffer_return, bytes_buffer, keysym_return, status_return)
    XIC ic;
    XKeyPressEvent *event;
    wchar_t *buffer_return;
    int wchars_buffer;
    KeySym *keysym_return;
    Status *status_return;
```

- `ic` Specifies the input context.
- `event` Specifies the key event to be used.
- `buffer_return` Returns a multibyte string or wide character string (if any) from the input method.
- `bytes_buffer` Specifies space available in return buffer.
- `wchars_buffer` Returns the KeySym computed from the event if this argument is not NULL.
- `status_return` Returns a value indicating what kind of data is returned.

The XmbLookupString and XwcLookupString functions return the string from the input method specified in the buffer_return argument. If no string is returned, the buffer_return argument is unchanged.

The KeySym into which the KeyCode from the event was mapped is returned in the keysym_return argument if it is non-NULL and the status_return argument indicates that a KeySym was returned. If both a string and a KeySym are returned, the KeySym value does not necessarily correspond to the string returned.

Note that XmbLookupString returns the length of the string in bytes and that XwcLookupString returns the length of the string in characters.

XmbLookupString and XwcLookupString return text in the encoding of the locale bound to the input method of the specified input context.

Note that each string returned by XmbLookupString and XwcLookupString begins in the initial state of the encoding of the locale (if the encoding of the locale is state-dependent).

In order to insure proper input processing, it is essential that the client pass only KeyPress events to XmbLookupString and XwcLookupString. Their behavior when a client passes a KeyRelease event is undefined.

Clients should check the status_return argument before using the other returned values. These two functions both return a value to status_return that indicates what has been returned in the other arguments. The possible values returned are:
XBufferOverflow
The input string to be returned is too large for the supplied buffer_return. The required size (XmbLookupString in bytes; XwcLookupString in characters) is returned as the value of the function, and the contents of buffer_return and keysym_return are not modified. The client should recall the function with the same event and a buffer of adequate size in order to obtain the string.

XLookupNone
No consistent input has been composed so far. The contents of buffer_return and keysym_return are not modified, and the function returns zero.

XLookupChars
Some input characters have been composed. They are placed in the buffer_return argument, and the string length is returned as the value of the function. The string is encoded in the locale bound to the input context. The contents of the keysym_return argument is not modified.

XLookupKeySym
A KeySym has been returned instead of a string and is returned in keysym_return. The contents of the buffer_return argument is not modified, and the function returns zero.

XLookupBoth
Both a KeySym and a string are returned; XLookupChars and XLookupKeySym occur simultaneously.

It does not make any difference if the input context passed as an argument to XmbLookupString and XwcLookupString is the one currently in possession of the focus or not. Input may have been composed within an input context before it lost the focus, and that input may be returned on subsequent calls to XmbLookupString or XwcLookupString, even though it does not have any more keyboard focus.

The input method architecture is transparent to the client. However, clients should respect a number of conventions in order to work properly. Clients must also be aware of possible effects of synchronization between input method and library in the case of a remote input server.

A well-behaved client (or toolkit) should first query the input method style. If the client cannot satisfy the requirements of the supported styles (in terms of geometry management or callbacks), it should negotiate with the user continuation of the program or raise an exception or error of some sort.

13.14.2. Synchronization Conventions
A KeyPress event with a KeyCode of zero is used exclusively as a signal that an input method has composed input which can be return by XmbLookupString or XwcLookupString. No other use is made of a KeyPress event with KeyCode of zero.

Such an event may be generated by either a front-end or a back-end input method in an implementation dependent manner. Some possible ways to generate this event include:

- A synthetic event sent by an input method server
- An artificial event created by a input method filter and pushed onto a client's event queue
- A KeyPress event whose KeyCode value is modified by an input method filter

When callback support is specified by client, input methods will not take action unless they explicitly called back the client and obtained no response (the callback is not specified or returned invalid data).
13.15. String Constants

The following symbols for string constants are defined in `<X11/Xlib.h>`. Although they are shown here with particular macro definitions, they may be implemented as macros, as global symbols, or as a mixture of the two. The string pointer value itself is not significant; clients must not assume that inequality of two values implies inequality of the actual string data.

```
#define XNVaNestedList "XNVaNestedList"
#define XNQueryInputStyle "queryInputStyle"
#define XNClientWindow "clientWindow"
#define XNInputStyle "inputStyle"
#define XNFocusWindow "focusWindow"
#define XNResourceName "resourceName"
#define XNResourceClass "resourceClass"
#define XNGeometryCallback "geometryCallback"
#define XNFilterEvents "filterEvents"
#define XNPreeditStartCallback "preeditStartCallback"
#define XNPreeditDoneCallback "preeditDoneCallback"
#define XNPreeditDrawCallback "preeditDrawCallback"
#define XNPreeditCaretCallback "preeditCaretCallback"
#define XNPreeditAttributes "preeditAttributes"
#define XNStatusStartCallback "statusStartCallback"
#define XNStatusDoneCallback "statusDoneCallback"
#define XNStatusDrawCallback "statusDrawCallback"
#define XNStatusAttributes "statusAttributes"
#define XNArea "area"
#define XNAreaNeeded "areaNeeded"
#define XNSpotLocation "spotLocation"
#define XNColorMap "colorMap"
#define XNStdColorMap "stdColorMap"
#define XNForeground "foreground"
#define XNBackground "background"
#define XNBackgroundPixmap "backgroundPixmap"
#define XNFontSet "fontSet"
#define XNLineSpace "lineSpace"
#define XNCursor "cursor"
```
Chapter 14

Inter-Client Communication Functions

The *Inter-Client Communication Conventions Manual*, hereafter referred to as the ICCCM, details the X Consortium approved conventions that govern inter-client communications. These conventions ensure peer-to-peer client cooperation in the use of selections, cut buffers, and shared resources as well as client cooperation with window and session managers. For further information, see the *Inter-Client Communication Conventions Manual*.

Xlib provides a number of standard properties and programming interfaces that are ICCCM compliant. The predefined atoms for some of these properties are defined in the `<Xlib/Xatom.h>` header file, where to avoid name conflicts with user symbols their `#define` name has an `XA_` prefix. For further information about atoms and properties, see section 4.3.

Xlib's selection and cut buffer mechanisms provide the primary programming interfaces by which peer client applications communicate with each other (see sections 4.5 and 16.6). The functions discussed in this chapter provide the primary programming interfaces by which client applications communicate with their window and session managers as well as share standard colormaps.

The standard properties that are of special interest for communicating with window and session managers are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WM_CLASS</td>
<td>STRING</td>
<td>8</td>
<td>Set by application programs to allow window and session managers to obtain the application's resources from the resource database.</td>
</tr>
<tr>
<td>WM_CLIENT_MACHINE</td>
<td>TEXT</td>
<td></td>
<td>The string name of the machine on which the client application is running.</td>
</tr>
<tr>
<td>WM_COLORMAP_WINDOWS</td>
<td>WINDOW</td>
<td>32</td>
<td>The list of window IDs that may need a different colormap than that of their top-level window.</td>
</tr>
<tr>
<td>WM_COMMAND</td>
<td>TEXT</td>
<td></td>
<td>The command and arguments, null-separated, used to invoke the application.</td>
</tr>
<tr>
<td>WM_HINTS</td>
<td>WM_HINTS</td>
<td>32</td>
<td>Additional hints set by the client for use by the window manager. The C type of this property is XWMHints.</td>
</tr>
<tr>
<td>WM_ICON_NAME</td>
<td>TEXT</td>
<td></td>
<td>The name to be used in an icon.</td>
</tr>
<tr>
<td>WM_ICON_SIZE</td>
<td>WM_ICON_SIZE</td>
<td>32</td>
<td>The window manager may set this property on the root window to specify the icon sizes it supports. The C type of this property is XIconSize.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WM_NAME</td>
<td>TEXT</td>
<td></td>
<td>The name of the application.</td>
</tr>
<tr>
<td>WM_NORMAL_HINTS</td>
<td>WM_SIZE_HINTS</td>
<td>32</td>
<td>Size hints for a window in its normal state. The C type of this property is XSizeHints.</td>
</tr>
<tr>
<td>WM_PROTOCOLS</td>
<td>ATOM</td>
<td>32</td>
<td>List of atoms that identify the communications protocols between the client and window manager in which the client is willing to participate.</td>
</tr>
<tr>
<td>WM_STATE</td>
<td>WM_STATE</td>
<td>32</td>
<td>Intended for communication between window and session managers only.</td>
</tr>
<tr>
<td>WM_TRANSIENT_FOR</td>
<td>WINDOW</td>
<td>32</td>
<td>Set by application programs to indicate to the window manager that a transient top-level window, such as a dialog box.</td>
</tr>
</tbody>
</table>

The remainder of this chapter discusses:
- Client-to-window-manager communication
- Client-to-session-manager communication
- Standard colormaps

14.1. Client to Window Manager Communication

This section discusses how to:
- Manipulate top-level windows
- Convert string lists
- Set and read text properties
- Set and read the WM_NAME property
- Set and read the WM_ICON_NAME property
- Set and read the WM_HINTS property
- Set and read the WM_NORMAL_HINTS property
- Set and read the WM_CLASS property
- Set and read the WM_TRANSIENT_FOR property
- Set and read the WM_PROTOCOLS property
- Set and read the WM_COLORMAP_WINDOWS property
- Set and read the WM_ICON_SIZE property
- Use window manager convenience functions

14.1.1. Manipulating Top-Level Windows

Xlib provides functions that you can use to change the visibility or size of top-level windows (that is, those that were created as children of the root window). Note that the subwindows that you create are ignored by window managers. Therefore, you should use the basic window functions described in chapter 3 to manipulate your application's subwindows.
To request that a top-level window be iconified, use `XIconifyWindow`.

```c
Status XIconifyWindow(display, w, screen_number)
    Display *display;
    Window w;
    int screen_number;

display    Specifies the connection to the X server.
w        Specifies the window.
screen_number        Specifies the appropriate screen number on the host server.
```

The `XIconifyWindow` function sends a WM_CHANGE_STATE ClientMessage event with a format of 32 and a first data element of IconicState (as described in section 4.1.4 of the Inter-Client Communication Conventions Manual) and a window of w to the root window of the specified screen with an event mask set to SubstructureNotifyMask | SubstructureRedirectMask. Window managers may elect to receive this message and if the window is in its normal state, may treat it as a request to change the window's state from normal to iconic. If the WM_CHANGE_STATE property cannot be interned, `XIconifyWindow` does not send a message and returns a zero status. It returns a nonzero status if the client message is sent successfully; otherwise, it returns a zero status.

To request that a top-level window be withdrawn, use `XWithdrawWindow`.

```c
Status XWithdrawWindow(display, w, screen_number)
    Display *display,
    Window w;
    int screen_number;

display    Specifies the connection to the X server.
w        Specifies the window.
screen_number        Specifies the appropriate screen number on the host server.
```

The `XWithdrawWindow` function unmaps the specified window and sends a synthetic UnmapNotify event to the root window of the specified screen. Window managers may elect to receive this message and may treat it as a request to change the window's state to withdrawn. When a window is in the withdrawn state, neither its normal nor its iconic representations is visible. It returns a nonzero status if the UnmapNotify event is successfully sent; otherwise, it returns a zero status.

`XWithdrawWindow` can generate a BadWindow error.

To request that a top-level window be reconfigured, use `XReconfigureWMWindow`.

```c
Status XReconfigureWMWindow(display, w, screen_number, value_mask, values)
    Display *display;
    Window w;
    int screen_number;
    unsigned int value_mask;
    XWindowChanges *values;

display    Specifies the connection to the X server.
w        Specifies the window.
screen_number        Specifies the appropriate screen number on the host server.
value_mask        Specifies which values are to be set using information in the values structure.
This mask is the bitwise inclusive OR of the valid configure window values bits.
```

262
values

Specifies the XWindowChanges structure.

The XReconfigureWMWindow function issues a ConfigureWindow request on the specified top-level window. If the stacking mode is changed and the request fails with a BadMatch error, the error is trapped by Xlib and a synthetic ConfigureRequestEvent containing the same configuration parameters is sent to the root of the specified window. Window managers may elect to receive this event and treat it as a request to reconfigure the indicated window. It returns a nonzero status if the request or event is successfully sent; otherwise, it returns a zero status.

XReconfigureWMWindow can generate BadValue and BadWindow errors.

14.1.2. Converting String Lists

Many of the text properties allow a variety of types and formats. Because the data stored in these properties are not simple null-terminated strings, a XTextProperty structure is used to describe the encoding, type, and length of the text as well as its value. The XTextProperty structure contains:

typedef struct {
    unsigned char *value; /* property data */
    Atom encoding; /* type of property */
    int format; /* 8, 16, or 32 */
    unsigned long nitems; /* number of items in value */
} XTextProperty;

Xlib provides functions to convert localized text to or from encodings which support the inter-client communication conventions for text. In addition, functions are provided for converting between lists of pointers to character strings and text properties in the STRING encoding.

The functions for localized text return a signed integer error status which encodes Success as zero, specific error conditions as negative numbers, and partial conversion as a count of unconvertible characters.

#define XNoMemory -1
#define XLocaleNotSupported -2
#define XConverterNotFound -3

typedef enum {
    XStringStyle, /* STRING */
    XCompoundTextStyle, /* COMPOUND_TEXT */
    XTextStyle, /* text in owner’s encoding (current locale) */
    XStdICCTextStyle /* STRING, else COMPOUND_TEXT */
} XICCEncodingStyle;

To convert a list of text strings to an XTextProperty structure, use XmbTextListToTextProperty or XwcTextListToTextProperty.

int XmbTextListToTextProperty(display, list, count, style, text_prop_return)

    Display *display;
    char **list;
    int count;
    XICCEncodingStyle style;
    XTextProperty *text_prop_return;
int XwcTextListToTextProperty(display, list, count, style, text_prop_return)
    Display *display;
    wchar_t **list;
    int count;
    XICCEncodingStyle style;
    XTextProperty *text_prop_return;

    display     Specifies the connection to the X server.
    list        Specifies a list of null-terminated character strings.
    count       Specifies the number of strings specified.
    style       Specifies the manner in which the property is encoded.
    text_prop_return Returns the XTextProperty structure.

The XmbTextListToTextProperty and XwcTextListToTextProperty functions set the specified XTextProperty value to a set of null-separated elements representing the concatenation of the specified list of null-terminated text strings. A final terminating null is stored at the end of the value field of text_prop_return but is not included in the nitems member.

The functions set the encoding field of text_prop_return to an Atom for the specified display naming the encoding determined by the specified style, and convert the specified text list to this encoding for storage in the text_prop_return value field. If the style XStringStyle or XCompoundTextStyle is specified, this encoding is "STRING" or "COMPOUND_TEXT", respectively. If the style XTextStyle is specified, this encoding is the encoding of the current locale. If the style XStdICCTextStyle is specified, this encoding is "STRING" if the text is fully convertible to STRING, else "COMPOUND_TEXT".

If insufficient memory is available for the new value string, the functions return XNoMemory. If the current locale is not supported, the functions return XLocaleNotSupported. In both of these error cases, the functions do not set text_prop_return.

To determine if the functions are guaranteed not to return XLocaleNotSupported, use XSupportsLocale.

If the supplied text is not fully convertible to the specified encoding, the functions return the number of unconvertible characters. Each unconvertible character is converted to an implementation-defined and encoding-specific default string. Otherwise, the functions return Success. Note that full convertibility to all styles except XStringStyle is guaranteed.

To free the storage for the value field, use XFree.

To obtain a list of text strings from an XTextProperty structure, use XmbTextPropertyToTextList or XwcTextPropertyToTextList.

int XmbTextPropertyToTextList(display, text_prop, list_return, count_return)
    Display *display;
    XTextProperty *text_prop;
    char ***list_return;
    int *count_return;

int XwcTextPropertyToTextList(display, text_prop, list_return, count_return)
    Display *display;
    XTextProperty *text_prop;
    wchar_t ***list_return;
    int *count_return;

display     Specifies the connection to the X server.
        text_prop     Specifies the XTextProperty structure to be used.
        list_return    Returns a list of null-terminated character strings.
count_result  Returns the number of strings.

The XmbTextPropertyToTextList and XwcTextPropertyToTextList functions return a list of text strings in the current locale representing the null-separated elements of the specified XTTextProperty structure. The data in text_prop must be format 8.

Multiple elements of the property (for example, the strings in a disjoint text selection) are separated by a null byte. The contents of the property are not required to be null-terminated; any terminating null should not be included in text_prop.nitems.

If insufficient memory is available for the list and its elements, XmbTextPropertyToTextList and XwcTextPropertyToTextList return XNoMemory. If the current locale is not supported, the functions return XLocaleNotSupported. Otherwise, if the encoding field of text_prop is not convertible to the encoding of the current locale, the functions return XConverterNotFound. For supported locales, existence of a converter from COMPOUND_TEXT, STRING, or the encoding of the current locale is guaranteed if XSUPPORTSlocale returns True for the current locale (but the actual text may contain unconvertible characters.) Conversion of other encodings is implementation-dependent. In all of these error cases, the functions do not set any return values.

Otherwise, XmbTextPropertyToTextList and XwcTextPropertyToTextList return the list of null-terminated text strings to list_return, and the number of text strings to count_return.

If the value field of text_prop is not fully convertible to the encoding of the current locale, the functions return the number of unconvertible characters. Each unconvertible character is converted to a string in the current locale that is specific to the current locale. To obtain the value of this string, use XDefaultString. Otherwise, XmbTextPropertyToTextList and XwcTextPropertyToTextList return Success.

To free the storage for the list and its contents returned by XmbTextPropertyToTextList, use XFreeStringList. To free the storage for the list and its contents returned by XwcTextPropertyToTextList, use XwcFreeStringList.

To free the in-memory data associated with the specified wide character string list, use XwcFreeStringList.

void XwcFreeStringList(list)
    wchar_t **list;

list  Specifies the list of strings to be freed.

The XwcFreeStringList function frees memory allocated by XwcTextPropertyToTextList.

To obtain the default string for text conversion in the current locale, use XDefaultString.

char *XDefaultString()

The XDefaultString function returns the default string used by Xlib for text conversion (for example, in XmbTextListToTextProperty). The default string is the string in the current locale which is output when an unconvertible character is found during text conversion. If the string returned by XDefaultString is the empty string (""), no character is output in the converted text. XDefaultString does not return NULL.

The string returned by XDefaultString is independent of the default string for text drawing; see XCreateFontSet to obtain the default string for an XFontSet.

The behavior when an invalid codepoint is supplied to any Xlib function is undefined.

The returned string is null-terminated. It is owned by Xlib and should not be modified or freed by the client. It may be freed after the current locale is changed. Until freed, it will not be modified by Xlib.
To set the specified list of strings in the STRING encoding to a XTextProperty structure, use XStringListToTextProperty.

Status XStringListToTextProperty(list, count, text_prop_return)

    char **list;
    int count;
    XTextProperty *text_prop_return;

list          Specifies a list of null-terminated character strings.

count         Specifies the number of strings.

text_prop_return          Returns the XTextProperty structure.

The XStringListToTextProperty function sets the specified XTextProperty to be of type STRING (format 8) with a value representing the concatenation of the specified list of null-separated character strings. An extra null byte (which is not included in the nitems member) is stored at the end of the value field of text_prop_return. The strings are assumed (without verification) to be in the STRING encoding. If insufficient memory is available for the new value string, XStringListToTextProperty does not set any fields in the XTextProperty structure and returns a zero status. Otherwise, it returns a nonzero status. To free the storage for the value field, use XFree.

To obtain a list of strings from a specified XTextProperty structure in the STRING encoding, use XTextPropertyToStringList.

Status XTextPropertyToStringList(text_prop, list_return, count_return)

    XTextProperty *text_prop;
    char ***list_return;
    int *count_return;

    text_prop          Specifies the XTextProperty structure to be used.

    list_return        Returns a list of null-terminated character strings.

    count_return       Returns the number of strings.

The XTextPropertyToStringList function returns a list of strings representing the null-separated elements of the specified XTextProperty structure. The data in text_prop must be of type STRING and format 8. Multiple elements of the property (for example, the strings in a disjoint text selection) are separated by NULL (encoding 0). The contents of the property are not null-terminated. If insufficient memory is available for the list and its elements, XTextPropertyToStringList sets no return values and returns a zero status. Otherwise, it returns a nonzero status. To free the storage for the list and its contents, use XFreeStringList.

To free the in-memory data associated with the specified string list, use XFreeStringList.

void XFreeStringList(list)

    char **list;

list     Specifies the list of strings to be freed.

The XFreeStringList function releases memory allocated by XmbTextPropertyToTextList and XTextPropertyToStringList, and the missing charset list allocated by XCreateFontSet.

14.1.3. Setting and Reading Text Properties

Xlib provides two functions that you can use to set and read the text properties for a given window. You can use these functions to set and read those properties of type TEXT (WM_NAME, WM_ICON_NAME, WM_COMMAND, and WM_CLIENT_MACHINE). In addition, Xlib provides separate convenience functions that you can use to set each of these properties. For further information about these convenience functions, see sections 14.1.4,
To set one of a window’s text properties, use XSetTextProperty.

```c
void XSetTextProperty(display, w, text_prop, property)
    Display *display;
    Window w;
    XTextProperty *text_prop;
    Atom property;

    display Specifies the connection to the X server.
    w Specifies the window.
    text_prop Specifies the XTextProperty structure to be used.
    property Specifies the property name.
```

The XSetTextProperty function replaces the existing specified property for the named window with the data, type, format, and number of items determined by the value field, the encoding field, the format field, and the nitems field, respectively, of the specified XTextProperty structure. If the property does not already exist, XSetTextProperty sets it for the specified window.

XSetTextProperty can generate BadAlloc, BadAtom, BadValue, and BadWindow errors.

To read one of a window’s text properties, use XGetTextProperty.

```c
Status XGetTextProperty(display, w, text_prop_return, property)
    Display *display;
    Window w;
    XTextProperty *text_prop_return;
    Atom property;

    display Specifies the connection to the X server.
    w Specifies the window.
    text_prop_return Returns the XTextProperty structure.
    property Specifies the property name.
```

The XGetTextProperty function reads the specified property from the window and stores the data in the returned XTextProperty structure. It stores the data in the value field, the type of the data in the encoding field, the format of the data in the format field, and the number of items of data in the nitems field. An extra byte containing null (which is not included in the nitems member) is stored at the end of the value field of text_prop_return. The particular interpretation of the property’s encoding and data as “text” is left to the calling application. If the specified property does not exist on the window, XGetTextProperty sets the value field to NULL, the encoding field to None, the format field to zero, and the nitems field to zero.

If it was able to read and store the data in the XTextProperty structure, XGetTextProperty returns a nonzero status; otherwise, it returns a zero status.

XGetTextProperty can generate BadAtom and BadWindow errors.

14.1.4. Setting and Reading the WM_NAME Property

Xlib provides convenience functions that you can use to set and read the WM_NAME property for a given window.

To set a window’s WM_NAME property with the supplied convenience function, use XSetWMName.
void XSetWMName(display, w, text_prop)
  Display *display;
  Window w;
  XTextProperty *text_prop;

display  Specifies the connection to the X server.
w    Specifies the window.
text_prop  Specifies the XTextProperty structure to be used.
The XSetWMName convenience function calls XSetTextProperty to set the WM_NAME property.

To read a window’s WM_NAME property with the supplied convenience function, use XGetWMName.
Status XGetWMName(display, w, text_prop_return)
  Display *display;
  Window w;
  XTextProperty *text_prop_return;

display  Specifies the connection to the X server.
w    Specifies the window.
text_prop_return  Returns the XTextProperty structure.
The XGetWMName convenience function calls XGetTextProperty to obtain the WM_NAME property. It returns nonzero status on success; otherwise it returns a zero status.
The following two functions have been superseded by XSetWMName and XGetWMName, respectively. You can use these additional convenience functions for window names that are encoded as STRING properties.

To assign a name to a window, use XStoreName.
XStoreName(display, w, window_name)
  Display *display;
  Window w;
  char *window_name;

display  Specifies the connection to the X server.
w    Specifies the window.
window_name  Specifies the window name, which should be a null-terminated string.
The XStoreName function assigns the name passed to window_name to the specified window. A window manager can display the window name in some prominent place, such as the title bar, to allow users to identify windows easily. Some window managers may display a window’s name in the window’s icon, although they are encouraged to use the window’s icon name if one is provided by the application. If the string is not in the Host Portable Character Encoding the result is implementation dependent.
XStoreName can generate BadAlloc and BadWindow errors.

To get the name of a window, use XFetchName.
Status XFetchName(display, w, window_name_return)
  Display *display;
  Window w;
  char **window_name_return;

268
display Specifies the connection to the X server.
w Specifies the window.

window_name_return Returns the window name, which is a null-terminated string.

The XFetchnName function returns the name of the specified window. If it succeeds, it returns nonzero; otherwise, no name has been set for the window, and it returns zero. If the WM_NAME property has not been set for this window, XFetchnName sets window_name_return to NULL. If the data returned by the server is in the Latin Portable Character Encoding, then the returned string is in the Host Portable Character Encoding. Otherwise, the result is implementation dependent. When finished with it, a client must free the window name string using XFree.

XFetchnName can generate a BadWindow error.

14.1.5. Setting and Reading the WM_ICON_NAME Property

Xlib provides convenience functions that you can use to set and read the WM_ICON_NAME property for a given window.

To set a window’s WM_ICON_NAME property, use XSetWMIconName.

void XSetWMIconName (display, w, text_prop)
    Display *display;
    Window w;
    XTextProperty *text_prop;

display Specifies the connection to the X server.
w Specifies the window.
text_prop Specifies the XTextProperty structure to be used.

The XSetWMIconName convenience function calls XSetTextProperty to set the WM_ICON_NAME property.

To read a window’s WM_ICON_NAME property, use XGetWMIconName.

Status XGetWMIconName (display, w, text_prop_return)
    Display *display;
    Window w;
    XTextProperty *text_prop_return;

display Specifies the connection to the X server.
w Specifies the window.
text_prop_return Returns the XTextProperty structure.

The XGetWMIconName convenience function calls XGetTextProperty to obtain the WM_ICON_NAME property. It returns nonzero status on success; otherwise it returns a zero status.

The next two functions have been superseded by XSetWMIconName and XGetWMIconName, respectively. You can use these additional convenience functions for window names that are encoded as STRING properties.

To set the name to be displayed in a window’s icon, use XSetIconName.
XSetIconName(display, w, icon_name)
    Display *display;
    Window w;
    char *icon_name;

    display    Specifies the connection to the X server.
    w          Specifies the window.
    icon_name  Specifies the icon name, which should be a null-terminated string.

If the string is not in the Host Portable Character Encoding the result is implementation dependent. XSetIconName can generate BadAlloc and BadWindow errors.

To get the name a window wants displayed in its icon, use XGetIconName.
Status XGetIconName(display, w, icon_name_return)
    Display *display;
    Window w;
    char **icon_name_return;

    display    Specifies the connection to the X server.
    w          Specifies the window.
    icon_name_return     Returns the window’s icon name, which is a null-terminated string.

The XGetIconName function returns the name to be displayed in the specified window’s icon. If it succeeds, it returns nonzero; otherwise, if no icon name has been set for the window, it returns zero. If you never assigned a name to the window, XGetIconName sets icon_name_return to NULL. If the data returned by the server is in the Latin Portable Character Encoding, then the returned string is in the Host Portable Character Encoding. Otherwise, the result is implementation dependent. When finished with it, a client must free the icon name string using XFree.
XGetIconName can generate a BadWindow error.

14.1.6. Setting and Reading the WM_HINTS Property
Xlib provides functions that you can use to set and read the WM_HINTS property for a given window. These functions use the flags and the XWMHints structure, as defined in the <X11/Xutil.h> header file.

To allocate an XWMHints structure, use XAllocWMHints.
XWMHints *XAllocWMHints()

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

The XWMHints structure contains:
/* Window manager hints mask bits */

#define InputHint      (1L << 0)
#define StateHint      (1L << 1)
#define IconPixmapHint (1L << 2)
#define IconWindowHint (1L << 3)
#define IconPositionHint (1L << 4)
#define IconMaskHint   (1L << 5)
#define WindowGroupHint (1L << 6)
#define AllHints (InputHintStateHintIconPixmapHint IconWindowHintIconPositionHint IconMaskHint WindowGroupHint)

/* Values */

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

The input member is used to communicate to the window manager the input focus model used by the application. Applications that expect input but never explicitly set focus to any of their subwindows (that is, use the push model of focus management), such as X Version 10 style applications that use real-estate driven focus, should set this member to True. Similarly, applications that set input focus to their subwindows only when it is given to their top-level window by a window manager should also set this member to True. Applications that manage their own input focus by explicitly setting focus to one of their subwindows whenever they want keyboard input (that is, use the pull model of focus management) should set this member to False. Applications that never expect any keyboard input also should set this member to False.

Pull model window managers should make it possible for push model applications to get input by setting input focus to the top-level windows of applications whose input member is True. Push model window managers should make sure that pull model applications do not break them by resetting input focus to PointerRoot when it is appropriate (for example, whenever an application whose input member is False sets input focus to one of its subwindows).

The definitions for the initial_state flag are:

#define WithdrawnState 0
#define NormalState 1 /* most applications start this way */
#define IconicState 3 /* application wants to start as an icon */

The icon_mask specifies which pixels of the icon_pixmap should be used as the icon. This allows for nonrectangular icons. Both icon_pixmap and icon_mask must be bitmaps. The icon_window lets an application provide a window for use as an icon for window managers that support such use. The window_group lets you specify that this window belongs to a group of other windows. For example, if a single application manipulates multiple top-level windows, this allows you to provide enough information that a window manager can iconify all of the windows rather than just the one window.

To set a window’s WM_HINTS property, use XSetWMHints.
XSetWMHints(display, w, wmhints)
Display *display;
Window w;
XWMHints *wmhints;

display Specifies the connection to the X server.
w Specifies the window.
wmhints Specifies the XWMHints structure to be used.

The XSetWMHints function sets the window manager hints that include icon information and location, the initial state of the window, and whether the application relies on the window manager to get keyboard input.

XSetWMHints can generate BadAlloc and BadWindow errors.

To read a window’s WM_HINTS property, use XGetWMHints.
XWMHints *XGetWMHints(display, w)
Display *display;
Window w;

display Specifies the connection to the X server.
w Specifies the window.

The XGetWMHints function reads the window manager hints and returns NULL if no WM_HINTS property was set on the window or returns a pointer to a XWMHints structure if it succeeds. When finished with the data, free the space used for it by calling XFree.

XGetWMHints can generate a BadWindow error.

14.1.7. Setting and Reading the WM_NORMAL_HINTS Property

Xlib provides functions that you can use to set or read the WM_NORMAL_HINTS property for a given window. The functions use the flags and the XSizeHints structure, as defined in the <X11/Xutil.h> header file.

To allocate an XSizeHints structure, use XAllocSizeHints.
XSizeHints *XAllocSizeHints()

The XAllocSizeHints function allocates and returns a pointer to a XSizeHints structure. Note that all fields in the XSizeHints structure are initially set to zero. If insufficient memory is available, XAllocSizeHints returns NULL. To free the memory allocated to this structure, use XFree.

The XSizeHints structure contains:
/* Size hints mask bits * /

#define USPosition (1L << 0) /* user specified x, y */
#define USSize (1L << 1) /* user specified width, height */
#define PPosition (1L << 2) /* program specified position */
#define PSize (1L << 3) /* program specified size */
#define PMinSize (1L << 4) /* program specified minimum size */
#define PMaxSize (1L << 5) /* program specified maximum size */
#define PResizeInc (1L << 6) /* program specified resize increments */
#define PAspect (1L << 7) /* program specified min and max aspect ratio */
#define PBaseSize (1L << 8)

272
#define PWinGravity (1L << 9)
#define PAHHints (PPositionlPSizelPMinSize1
               PMaxSize1PResizeInc1PAспект)

/* Values */

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

The x, y, width, and height members are now obsolete and are left solely for compatibility reasons. The min_width and min_height members specify the minimum window size that still allows the application to be useful. The max_width and max_height members specify the maximum window size. The width_inc and height_inc members define an arithmetic progression of sizes (minimum to maximum) into which the window prefers to be resized. The min_aspect and max_aspect members are expressed as ratios of x and y, and they allow an application to specify the range of aspect ratios it prefers. The base_width and base_height members define the desired size of the window. The window manager will interpret the position of the window and its border width to position the point of the outer rectangle of the overall window specified by the win_gravity member. The outer rectangle of the window includes any borders or decorations supplied by the window manager. In other words, if the window manager decides to place the window where the client asked, the position on the parent window’s border named by the win_gravity will be placed where the client window would have been placed in the absence of a window manager.

Note that use of the PAHHints macro is highly discouraged.

To set a window’s WM_NORMAL_HINTS property, use XSetWMNormalHints.

void XSetWMNormalHints(display, w, hints)
    Display *display;
    Window w;
    XSizeHints *hints;

display Specifies the connection to the X server.
w Specifies the window.
hints Specifies the size hints for the window in its normal state.

The XSetWMNormalHints function replaces the size hints for the WM_NORMAL_HINTS property on the specified window. If the property does not already exist, XSetWMNormalHints sets the size hints for the WM_NORMAL_HINTS property on the specified window. The property is stored with a type of WM_SIZE_HINTS and a format of 32.

XSetWMNormalHints can generate BadAlloc and BadWindow errors.
To read a window’s WM_NORMAL_HINTS property, use XGetWMNormalHints.

```c
Status XGetWMNormalHints (Display *display, Window w, XSizeHints *hints_return, long *supplied_return)
```

- `display` Specifies the connection to the X server.
- `w` Specifies the window.
- `hints_return` Returns the size hints for the window in its normal state.
- `supplied_return` Returns the hints that were supplied by the user.

The XGetWMNormalHints function returns the size hints stored in the WM_NORMAL_HINTS property on the specified window. If the property is of type WM_SIZE_HINTS, is of format 32, and is long enough to contain either an old (pre-ICCCM) or new size hints structure, XGetWMNormalHints sets the various fields of the XSizeHints structure, sets the supplied_return argument to the list of fields that were supplied by the user (whether or not they contained defined values), and returns a nonzero status. Otherwise, it returns a zero status.

If XGetWMNormalHints returns successfully and a pre-ICCCM size hints property is read, the supplied_return argument will contain the following bits:

```
(USPositionl USize1 PPosition1 PSize1 PMinSize1
 PMaxSize1 PResizeInd PAspect)
```

If the property is large enough to contain the base size and window gravity fields as well, the supplied_return argument will also contain the following bits:

```
PBaseSize PWinGravity
```

XGetWMNormalHints can generate a BadWindow error.

To set a window’s WM_SIZE_HINTS property, use XSetWMSizeHints.

```c
void XSetWMSizeHints (Display *display, Window w, XSizeHints *hints, Atom property)
```

- `display` Specifies the connection to the X server.
- `w` Specifies the window.
- `hints` Specifies the XSizeHints structure to be used.
- `property` Specifies the property name.

The XSetWMSizeHints function replaces the size hints for the specified property on the named window. If the specified property does not already exist, XSetWMSizeHints sets the size hints for the specified property on the named window. The property is stored with a type of WM_SIZE_HINTS and a format of 32. To set a window’s normal size hints, you can use the XSetWMNormalHints function.

XSetWMSizeHints can generate BadAlloc, BadAtom, and BadWindow errors.

To read a window’s WM_SIZE_HINTS property, use XGetWMSizeHints.
Status XGetWMSizeHints(display, w, hints_return, supplied_return, property)
   Display *display;
   Window w;
   XSizeHints *hints_return;
   long *supplied_return;
   Atom property;

display   Specifies the connection to the X server.
w   Specifies the window.
hints_return Returns the XSizeHints structure.
supplied_return Returns the hints that were supplied by the user.
property   Specifies the property name.

The XGetWMSizeHints function returns the size hints stored in the specified property on the
named window. If the property is of type WM_SIZE_HINTS, is of format 32, and is long
enough to contain either an old (pre-ICCCM) or new size hints structure, XGetWMSizeHints
sets the various fields of the XSizeHints structure, sets the supplied_return argument to the list
of fields that were supplied by the user (whether or not they contained defined values), and
returns a nonzero status. Otherwise, it returns a zero status. To get a window’s normal size
hints, you can use the XGetWMNormalHints function.

If XGetWMSizeHints returns successfully and a pre-ICCCM size hints property is read, the
supplied_return argument will contain the following bits:

   (USPositionl USSizel PPositionl PSize\ PMinSizel
   PMaxSizel PReszieIn\ PAspect)

If the property is large enough to contain the base size and window gravity fields as well, the
supplied_return argument will also contain the following bits:

   PBaseSize\ PWinGravity

XGetWMSizeHints can generate BadAtom and BadWindow errors.

14.1.8. Setting and Reading the WM_CLASS Property

Xlib provides functions that you can use to set and get the WM_CLASS property for a given
window. These functions use the XClassHint structure, which is defined in the
<X11/Xutil.h> header file.

To allocate an XClassHint structure, use XAllocClassHint.

XClassHint *XAllocClassHint()

The XAllocClassHint function allocates and returns a pointer to a XClassHint structure.

The XClassHint structure contains:

typedef struct {
   char *res__name;
   char *res_class;
} XClassHint;

The res__name member contains the application name, and the res_class member contains the
application class. Note that the name set in this property may differ from the name set as
WM_NAME. That is, WM_NAME specifies what should be displayed in the title bar and,
therefore, can contain temporal information (for example, the name of a file currently in an editor's buffer). On the other hand, the name specified as part of WM_CLASS is the formal name of the application that should be used when retrieving the application's resources from the resource database.

To set a window's WM_CLASS property, use XSetClassHint.

\[
\text{XSetClassHint}(display, w, \text{class\_hints})
\]

- \text{display} Specifies the connection to the X server.
- \text{w} Specifies the window.
- \text{class\_hints} Specifies the XClassHint structure that is to be used.

The XSetClassHint function sets the class hint for the specified window. If the strings are not in the Host Portable Character Encoding the result is implementation dependent.

XSetClassHint can generate BadAlloc and BadWindow errors.

To read a window's WM_CLASS property, use XGetClassHint.

\[
\text{Status XGetClassHint}(display, w, \text{class\_hints\_return})
\]

- \text{display} Specifies the connection to the X server.
- \text{w} Specifies the window.
- \text{class\_hints\_return} Returns the XClassHint structure.

The XGetClassHint function returns the class hint of the specified window to the members of the supplied structure. If the data returned by the server is in the Latin Portable Character Encoding, then the returned strings are in the Host Portable Character Encoding. Otherwise, the result is implementation dependent. It returns nonzero status on success; otherwise it returns a zero status. To free res\_name and res\_class when finished with the strings, use XFree on each individually.

XGetClassHint can generate a BadWindow error.

14.1.9. Setting and Reading the WM_TRANSIENT_FOR Property

Xlib provides functions that you can use to set and read the WM_TRANSIENT_FOR property for a given window.

To set a window's WM_TRANSIENT_FOR property, use XSetTransientForHint.

\[
\text{XSetTransientForHint}(display, w, \text{prop\_window})
\]

- \text{display} Specifies the connection to the X server.
- \text{w} Specifies the window.
- \text{prop\_window} Specifies the window that the WM_TRANSIENT_FOR property is to be set to.
The `XSetTransientForHint` function sets the WM_TRANSIENT_FOR property of the specified window to the specified `prop_window`.

`XSetTransientForHint` can generate BadAlloc and BadWindow errors.

To read a window’s WM_TRANSIENT_FOR property, use `XGetTransientForHint`.

Status `XGetTransientForHint(display, w, prop_window_return)`

- `display`: Specifies the connection to the X server.
- `w`: Specifies the window.
- `prop_window_return`: Returns the WM_TRANSIENT_FOR property of the specified window.

The `XGetTransientForHint` function returns the WM_TRANSIENT_FOR property for the specified window. It returns nonzero status on success; otherwise it returns a zero status.

`XSetWMProtocols` can generate a BadWindow error.

14.1.10. Setting and Reading the WM_PROTOCOLS Property

`Xlib` provides functions that you can use to set and read the WM_PROTOCOLS property for a given window.

To set a window’s WM_PROTOCOLS property, use `XSetWMProtocols`.

Status `XSetWMProtocols(display, w, protocols, count)`

- `display`: Specifies the connection to the X server.
- `w`: Specifies the window.
- `protocols`: Specifies the list of protocols.
- `count`: Specifies the number of protocols in the list.

The `XSetWMProtocols` function replaces the WM_PROTOCOLS property on the specified window with the list of atoms specified by the protocols argument. If the property does not already exist, `XSetWMProtocols` sets the WM_PROTOCOLS property on the specified window to the list of atoms specified by the protocols argument. The property is stored with a type of ATOM and a format of 32. If it cannot intern the WM_PROTOCOLS atom, `XSetWMProtocols` returns a zero status. Otherwise, it returns a nonzero status.

`XSetWMProtocols` can generate BadAlloc and BadWindow errors.

To read a window’s WM_PROTOCOLS property, use `XGetWMProtocols`.

Status `XGetWMProtocols(display, w, protocols_return, count_return)`

- `display`: Specifies the connection to the X server.
- `w`: Specifies the window.
w Specifies the window.
protocols_return Returns the list of protocols.
count_return Returns the number of protocols in the list.

The XGetWMProtocols function returns the list of atoms stored in the WM_PROTOCOLS property on the specified window. These atoms describe window manager protocols in which the owner of this window is willing to participate. If the property exists, is of type ATOM, is of format 32, and the atom WM_PROTOCOLS can be interned, XGetWMProtocols sets the protocols_return argument to a list of atoms, sets the count_return argument to the number of elements in the list, and returns a nonzero status. Otherwise, it sets neither of the return arguments and returns a zero status. To release the list of atoms, use XFree.

XGetWMProtocols can generate a BadWindow error.

14.1.11. Setting and Reading the WM_COLORMAP_WINDOWS Property

Xlib provides functions that you can use to set and read the WM_COLORMAP_WINDOWS property for a given window.

To set a window’s WM_COLORMAP_WINDOWS property, use XSetWMColormapWindows.

Status XSetWMColormapWindows(display, w, colormap_windows, count)
    Display *display;
    Window w;
    Window *colormap_windows;
    int count;

display Specifies the connection to the X server.
w Specifies the window.
colormap_windows Specifies the list of windows.
count Specifies the number of windows in the list.

The XSetWMColormapWindows function replaces the WM_COLORMAP_WINDOWS property on the specified window with the list of windows specified by the colormap_windows argument. If the property does not already exist, XSetWMColormapWindows sets the WM_COLORMAP_WINDOWS property on the specified window to the list of windows specified by the colormap_windows argument. The property is stored with a type of WINDOW and a format of 32. If it cannot intern the WM_COLORMAP_WINDOWS atom, XSetWMColormapWindows returns a zero status. Otherwise, it returns a nonzero status.

XSetWMColormapWindows can generate BadAlloc and BadWindow errors.

To read a window’s WM_COLORMAP_WINDOWS property, use XGetWMColormapWindows.

Status XGetWMColormapWindows(display, w, colormap_windows_return, count_return)
    Display *display;
    Window w;
    Window **colormap_windows_return;
    int *count_return;

display Specifies the connection to the X server.
w Specifies the window.
colormap_windows_return Returns the list of windows.
count_return  Returns the number of windows in the list.

The XGetWMColormapWindows function returns the list of window identifiers stored in the
WM_COLORMAP_WINDOWS property on the specified window. These identifiers indicate
the colormaps that the window manager may need to install for this window. If the property
exists, is of type WINDOW, is of format 32, and the atom WM_COLORMAP_WINDOWS
can be interned, XGetWMColormapWindows sets the windows_return argument to a list of
window identifiers, sets the count_return argument to the number of elements in the list, and
returns a nonzero status. Otherwise, it sets neither of the return arguments and returns a zero
status. To release the list of window identifiers, use XFree.

XGetWMColormapWindows can generate a BadWindow error.

14.1.12. Setting and Reading the WM_ICON_SIZE Property
Xlib provides functions that you can use to set and read the WM_ICON_SIZE property for a
given window. These functions use the XIconSize structure, which is defined in the
<X11/Xutil.h> header file.

To allocate an XIconSize structure, use XAllocIconSize.

XIconSize *XAllocIconSize()

The XAllocIconSize function allocates and returns a pointer to a XIconSize structure. Note
that all fields in the XIconSize structure are initially set to zero. If insufficient memory is
available, XAllocIconSize returns NULL. To free the memory allocated to this structure, use
XFree.

The XIconSize structure contains:

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

The width_inc and height_inc members define an arithmetic progression of sizes (minimum to
maximum) that represent the supported icon sizes.

To set a window’s WM_ICON_SIZE property, use XSetIconSizes.

XSetIconSizes(display, w, size_list, count)

Display *display;
Window w;
XIconSize *size_list;
int count;

display  Specifies the connection to the X server.
w  Specifies the window.
size_list  Specifies the size list.
count  Specifies the number of items in the size list.

The XSetIconSizes function is used only by window managers to set the supported icon sizes.
XSetIconSizes can generate BadAlloc and BadWindow errors.

To read a window’s WM_ICON_SIZE property, use XGetIconSizes.
Status XGetIconSizes(display, w, size_list_return, count_return)
  Display *display;
  Window w;
  XIconSize **size_list_return;
  int *count_return;

  display    Specifies the connection to the X server.
  w          Specifies the window.
  size_list_return Returns the size list.
  count_return Returns the number of items in the size list.

The XGetIconSizes function returns zero if a window manager has not set icon sizes; otherwise, it return nonzero. XGetIconSizes should be called by an application that wants to find out what icon sizes would be most appreciated by the window manager under which the application is running. The application should then use XSetWMHints to supply the window manager with an icon pixmap or window in one of the supported sizes. To free the data allocated in size_list_return, use XFree.

XGetIconSizes can generate a BadWindow error.

14.1.13. Using Window Manager Convenience Functions

The XmbSetWMProperties function stores the standard set of window manager properties, with text properties in standard encodings for internationalized text communication. The standard window manager properties for a given window are WM_NAME, WM_ICON_NAME, WM_HINTS, WM_NORMAL_HINTS, WM_CLASS, WM_COMMAND, WM_CLIENT_MACHINE, and WM_LOCALE_NAME.

void XmbSetWMProperties(display, w, window_name, icon_name, argv, argc, normal_hints, wm_hints, class_hints)
  Display *display;
  Window w;
  char *window_name;
  char *icon_name;
  char *argv[];
  int argc;
  XSizeHints *normal_hints;
  XWMHints *wm_hints;
  XClassHint *class_hints;

  display    Specifies the connection to the X server.
  w          Specifies the window.
  window_name Specifies the window name, which should be a null-terminated string.
  icon_name  Specifies the icon name, which should be a null-terminated string.
  argv       Specifies the application’s argument list.
  argc       Specifies the number of arguments.
  hints      Specifies the size hints for the window in its normal state.
  wm_hints   Specifies the XWMHints structure to be used.
  class_hints Specifies the XClassHint structure to be used.

The XmbSetWMProperties convenience function provides a simple programming interface for setting those essential window properties that are used for communicating with other clients (particularly window and session managers).

If the window_name argument is non-NULL, XmbSetWMProperties sets the WM_NAME property. If the icon_name argument is non-NULL, XmbSetWMProperties sets the
WM_ICON_NAME property. The window_name and icon_name arguments are null-terminated strings in the encoding of the current locale. If the arguments can be fully converted to the STRING encoding, the properties are created with type "STRING"; otherwise, the arguments are converted to Compound Text, and the properties are created with type "COMPOUND_TEXT".

If the normal_hints argument is non-NULL, XmbSetWMProperties calls XSetWMNormalHints, which sets the WM_NORMAL_HINTS property (see section 14.1.7). If the wm_hints argument is non-NULL, XmbSetWMProperties calls XSetWMHints, which sets the WM_HINTS property (see section 14.1.6).

If the argv argument is non-NULL, XmbSetWMProperties sets the WM_COMMAND property from argv and argc. Note that an argc of 0 indicates a zero-length command.

The hostname of this machine is stored using XSetWMClientMachine (see section 14.2.2).

If the class_hints argument is non-NULL, XmbSetWMProperties sets the WM_CLASS property. If the res_name member in the XClassHint structure is set to the NULL pointer and the RESOURCE_NAME environment variable is set, the value of the environment variable is substituted for res_name. If the res_name member is NULL, the environment variable is not set, and argv and argv[0] are set, then the value of argv[0], stripped of any directory prefixes, is substituted for res_name.

It is assumed that the supplied class_hints.res_name and argv, the RESOURCE_NAME environment variable, and the hostname of this machine are in the encoding of the locale announced for the LC_CTYPE category. (On POSIX-compliant systems, the LC_CTYPE, else LANG environment variable). The corresponding WM_CLASS, WM_COMMAND, and WM_CLIENT_MACHINE properties are typed according to the local host locale announcer. No encoding conversion is performed prior to storage in the properties.

For clients that need to process the property text in a locale, XmbSetWMProperties sets the WM_LOCALE_NAME property to be the name of the current locale. The name is assumed to be in the Host Portable Character Encoding, and is converted to STRING for storage in the property.

XmbSetWMProperties can generate BadAlloc and BadWindow errors.

To set a window's standard window manager properties with strings in the STRING encoding, use XSetWMProperties. The standard window manager properties for a given window are WM_NAME, WM_ICON_NAME, WM_HINTS, WM_NORMAL_HINTS, WM_CLASS, WM_COMMAND, and WM_CLIENT_MACHINE.

void XSetWMProperties (Display *display, Window w, XTextProperty *window_name, icon_name, XTextProperty *icon_name, argv, argc, XSizeHints *normal_hints, WMHints *wm_hints, class_hints);
argc       Specifies the number of arguments.

normal_hints       Specifies the size hints for the window in its normal state.

wm_hints       Specifies the XWMHints structure to be used.

class_hints       Specifies the XClassHint structure to be used.

The XSetWMProperties convenience function provides a single programming interface for setting those essential window properties that are used for communicating with other clients (particularly window and session managers).

If the window_name argument is non-NULL, XSetWMProperties calls XSetWMName, which in turn, sets the WM_NAME property (see section 14.1.4). If the icon_name argument is non-NULL, XSetWMProperties calls XSetWMIconName, which sets the WM_ICON_NAME property (see section 14.1.5). If the argv argument is non-NULL, XSetWMProperties calls XSetCommand, which sets the WM_COMMAND property (see section 14.2.1). Note that an argc of zero is allowed to indicate a zero-length command. Note also that the hostname of this machine is stored using XSetWMClientMachine (see section 14.2.2).

If the normal_hints argument is non-NULL, XSetWMProperties calls XSetWMNormalHints, which sets the WM_NORMAL_HINTS property (see section 14.1.7). If the wm_hints argument is non-NULL, XSetWMProperties calls XSetWMHints, which sets the WM_HINTS property (see section 14.1.6).

If the class_hints argument is non-NULL, XSetWMProperties calls XSetClassHint, which sets the WM_CLASS property (see section 14.1.8). If the res_name member in the XClassHint structure is set to the NULL pointer and the RESOURCE_NAME environment variable is set, then the value of the environment variable is substituted for res_name. If the res_name member is NULL, the environment variable is not set, and argv and argv[0] are set, then the value of argv[0], stripped of any directory prefixes, is substituted for res_name.

XSetWMProperties can generate BadAlloc and BadWindow errors.

14.2. Client to Session Manager Communication

This section discusses how to:

• Set and read the WM_COMMAND property
• Set and read the WM_CLIENT_MACHINE property

14.2.1. Setting and Reading the WM_COMMAND Property

Xlib provides functions that you can use to set and read the WM_COMMAND property for a given window.

To set a window’s WM_COMMAND property, use XSetCommand.

XSetCommand(display, w, argv, argc)

Display *display;
Window w;
char **argv;
int argc;

display       Specifies the connection to the X server.
w       Specifies the window.
argv       Specifies the application’s argument list.
argc       Specifies the number of arguments.

The XSetCommand function sets the command and arguments used to invoke the application. (Typically, argv is the argv array of your main program.) If the strings are not in the Host Portable Character Encoding the result is implementation dependent.
XSetCommand can generate BadAlloc and BadWindow errors.

To read a window’s WM_COMMAND property, use XGetCommand.

Status XGetCommand(display, w, argv_return, argc_return)

Display *display;
Window w;
char ***argv_return;
int *argc_return;

display Specifies the connection to the X server.
w Specifies the window.
argv_return Returns the application’s argument list.
argc_return Returns the number of arguments returned.

The XGetCommand function reads the WM_COMMAND property from the specified window and returns a string list. If the WM_COMMAND property exists, it is of type STRING and format 8. If sufficient memory can be allocated to contain the string list, XGetCommand fills in the argv_return and argc_return arguments and returns a nonzero status. Otherwise, it returns a zero status. If the data returned by the server is in the Latin Portable Character Encoding, then the returned strings are in the Host Portable Character Encoding. Otherwise, the result is implementation dependent. To free the memory allocated to the string list, use XFreeStringList.

14.2.2. Setting and Reading the WM_CLIENT_MACHINE Property

Xlib provides functions that you can use to set and read the WM_CLIENT_MACHINE property for a given window.

To set a window’s WM_CLIENT_MACHINE property, use XSetWMClientMachine.

void XSetWMClientMachine(display, w, text_prop)

Display *display;
Window w;
XTextProperty *text_prop;

display Specifies the connection to the X server.
w Specifies the window.
text_prop Specifies the XTextProperty structure to be used.

The XSetWMClientMachine convenience function calls XSetTextProperty to set the WM_CLIENT_MACHINE property.

To read a window’s WM_CLIENT_MACHINE property, use XGetWMClientMachine.

Status XGetWMClientMachine(display, w, text_prop_return)

Display *display;
Window w;
XTextProperty *text_prop_return;

display Specifies the connection to the X server.
w Specifies the window.
text_prop_return Returns the XTextProperty structure.

The XGetWMClientMachine convenience function performs an XGetTextProperty on the WM_CLIENT_MACHINE property. It returns nonzero status on success; otherwise it returns a zero status.
14.3. Standard Colormaps

Applications with color palettes, smooth-shaded drawings, or digitized images demand large numbers of colors. In addition, these applications often require an efficient mapping from color triples to pixel values that display the appropriate colors.

As an example, consider a three-dimensional display program that wants to draw a smoothly shaded sphere. At each pixel in the image of the sphere, the program computes the intensity and color of light reflected back to the viewer. The result of each computation is a triple of RGB coefficients in the range 0.0 to 1.0. To draw the sphere, the program needs a colormap that provides a large range of uniformly distributed colors. The colormap should be arranged so that the program can convert its RGB triples into pixel values very quickly, because drawing the entire sphere requires many such conversions.

On many current workstations, the display is limited to 256 or fewer colors. Applications must allocate colors carefully, not only to make sure they cover the entire range they need but also to make use of as many of the available colors as possible. On a typical X display, many applications are active at once. Most workstations have only one hardware look-up table for colors, so only one application colormap can be installed at a given time. The application using the installed colormap is displayed correctly, and the other applications "go technicolor" and are displayed with false colors.

As another example, consider a user who is running an image processing program to display earth-resources data. The image processing program needs a colormap set up with 8 reds, 8 greens, and 4 blues, for a total of 256 colors. Because some colors are already in use in the default colormap, the image processing program allocates and installs a new colormap.

The user decides to alter some of the colors in the image by invoking a color palette program to mix and choose colors. The color palette program also needs a colormap with eight reds, eight greens, and four blues, so just like the image processing program, it must allocate and install a new colormap.

Because only one colormap can be installed at a time, the color palette may be displayed incorrectly whenever the image processing program is active. Conversely, whenever the palette program is active, the image may be displayed incorrectly. The user can never match or compare colors in the palette and image. Contention for colormap resources can be reduced if applications with similar color needs share colormaps.

The image processing program and the color palette program could share the same colormap if there existed a convention that described how the colormap was set up. Whenever either program was active, both would be displayed correctly.

The standard colormap properties define a set of commonly used colormaps. Applications that share these colormaps and conventions display true colors more often and provide a better interface to the user.

Standard colormaps allow applications to share commonly used color resources. This allows many applications to be displayed in true colors simultaneously, even when each application needs an entirely filled colormap.

Several standard colormaps are described in this section. Usually, a window manager creates these colormaps. Applications should use the standard colormaps if they already exist.

To allocate an XStandardColormap structure, use XAllocStandardColormap.

XStandardColormap *XAllocStandardColormap()

The XAllocStandardColormap function allocates and returns a pointer to a XStandardColormap structure. Note that all fields in the XStandardColormap structure are initially set to zero. If insufficient memory is available, XAllocStandardColormap returns NULL. To free the memory allocated to this structure, use XFree.

284
The \texttt{XStandardColormap} structure contains:

/* Hints */
#define ReleaseByFreeingColormap \ ((XID) 1L)

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

The colormap member is the colormap created by the \texttt{XCreateColormap} function. The \texttt{red_max}, \texttt{green_max}, and \texttt{blue_max} members give the maximum red, green, and blue values, respectively. Each color coefficient ranges from zero to its max, inclusive. For example, a common colormap allocation is 3/3/2 (3 planes for red, 3 planes for green, and 2 planes for blue). This colormap would have \texttt{red_max} = 7, \texttt{green_max} = 7, and \texttt{blue_max} = 3. An alternate allocation that uses only 216 colors is \texttt{red_max} = 5, \texttt{green_max} = 5, and \texttt{blue_max} = 5.

The \texttt{red_mult}, \texttt{green_mult}, and \texttt{blue_mult} members give the scale factors used to compose a full pixel value. (See the discussion of the \texttt{base_pixel} members for further information.) For a 3/3/2 allocation, \texttt{red_mult} might be 32, \texttt{green_mult} might be 4, and \texttt{blue_mult} might be 1. For a 6-colors-each allocation, \texttt{red_mult} might be 36, \texttt{green_mult} might be 6, and \texttt{blue_mult} might be 1.

The \texttt{base_pixel} member gives the base pixel value used to compose a full pixel value. Usually, the \texttt{base_pixel} is obtained from a call to the \texttt{XAllocColorPlanes} function. Given integer red, green, and blue coefficients in their appropriate ranges, one then can compute a corresponding pixel value by using the following expression:

\[(r \times \texttt{red_mult} + g \times \texttt{green_mult} + b \times \texttt{blue_mult} + \texttt{base_pixel}) \& 0xFFFFFFFF\]

For \texttt{GrayScale} colormaps, only the colormap, \texttt{red_max}, \texttt{red_mult}, and \texttt{base_pixel} members are defined. The other members are ignored. To compute a \texttt{GrayScale} pixel value, use the following expression:

\[(\texttt{gray} \times \texttt{red_mult} + \texttt{base_pixel}) \& 0xFFFFFFFF\]

Negative multipliers can be represented by converting the 2's complement representation of the multiplier into an unsigned long and storing the result in the appropriate \texttt{mult} field. The step of masking by 0xFFFFFFFF effectively converts the resulting positive multiplier into a negative one. The masking step will take place automatically on many machine architectures, depending on the size of the integer type used to do the computation.

The \texttt{visualid} member gives the ID number of the visual from which the colormap was created. The \texttt{killid} member gives a resource ID that indicates whether the cells held by this standard colormap are to be released by freeing the colormap ID or by calling the \texttt{XKillClient} function on the indicated resource. (Note that this method is necessary for allocating out of an existing colormap.)

The properties containing the \texttt{XStandardColormap} information have the type \texttt{RGB_COLOR_MAP}.
The remainder of this section discusses standard colormap properties and atoms as well as how
to manipulate standard colormaps.

14.3.1. Standard Colormap Properties and Atoms
Several standard colormaps are available. Each standard colormap is defined by a property, and
each such property is identified by an atom. The following list names the atoms and describes
the colormap associated with each one. The <X11/Xatom.h> header file contains the
definitions for each of the following atoms, which are prefixed with XA_.

RGB_DEFAULT_MAP
This atom names a property. The value of the property is an array of XStandardColor-
map structures. Each entry in the array describes an RGB subset of the default color
map for the Visual specified by visual_id.

Some applications only need a few RGB colors and may be able to allocate them from
the system default colormap. This is the ideal situation because the fewer colormaps that
are active in the system the more applications are displayed with correct colors at all
times.

A typical allocation for the RGB_DEFAULT_MAP on 8-plane displays is 6 reds, 6
greens, and 6 blues. This gives 216 uniformly distributed colors (6 intensities of 36
different hues) and still leaves 40 elements of a 256-element colormap available for
special-purpose colors for text, borders, and so on.

RGB_BEST_MAP
This atom names a property. The value of the property is an XStandardColormap.

The property defines the best RGB colormap available on the screen. (Of course, this is
a subjective evaluation.) Many image processing and three-dimensional applications need
to use all available colormap cells and to distribute as many perceptually distinct colors
as possible over those cells. This implies that there may be more green values available
than red, as well as more green or red than blue.

For an 8-plane PseudoColor visual, RGB_BEST_MAP is likely to be a 3/3/2 allocation.
For a 24-plane DirectColor visual, RGB_BEST_MAP is normally an 8/8/8 allocation.

RGB_RED_MAP
RGB_GREEN_MAP
RGB_BLUE_MAP
These atoms name properties. The value of each property is an XStandardColormap.

The properties define all-red, all-green, and all-blue colormaps, respectively. These maps
are used by applications that want to make color-separated images. For example, a user
might generate a full-color image on an 8-plane display both by rendering an image three
times (once with high color resolution in red, once with green, and once with blue) and
by multiply-exposing a single frame in a camera.

RGB_GRAY_MAP
This atom names a property. The value of the property is an XStandardColormap.

The property describes the best GrayScale colormap available on the screen. As previ¬
ously mentioned, only the colormap, red_max, red mult, and base_pixel members of the
XStandardColormap structure are used for GrayScale colormaps.

14.3.2. Setting and Obtaining Standard Colormaps
Xlib provides functions that you can use to set and obtain an XStandardColormap structure.

To set an XStandardColormap structure, use XSetRGBColormaps.
void XSetRGBColormaps(display, w, std_colormap, count, property)
    Display *display;
    Window w;
    XStandardColormap *std_colormap;
    int count;
    Atom property;

display    Specifies the connection to the X server.
w         Specifies the window.
std_colormap Specifications the XStandardColormap structure to be used.
count          Specifies the number of colormaps.
property       Specifies the property name.

The XSetRGBColormaps function replaces the RGB colormap definition in the specified property on the named window. If the property does not already exist, XSetRGBColormaps sets the RGB colormap definition in the specified property on the named window. The property is stored with a type of RGB_COLOR_MAP and a format of 32. Note that it is the caller’s responsibility to honor the ICCCM restriction that only RGB_DEFAULT_MAP contain more than one definition.

The XSetRGBColormaps function usually is only used by window or session managers. To create a standard colormap, follow this procedure:
1. Open a new connection to the same server.
2. Grab the server.
3. See if the property is on the property list of the root window for the screen.
4. If the desired property is not present:
   • Create a colormap (unless using the default colormap of the screen).
   • Determine the color characteristics of the visual.
   • Call XAllocColorPlanes or XAllocColorCells to allocate cells in the colormap.
   • Call XStoreColors to store appropriate color values in the colormap.
   • Fill in the descriptive members in the XStandardColormap structure.
   • Attach the property to the root window.
   • Use XSetCloseDownMode to make the resource permanent.
5. Ungrab the server.

XSetRGBColormaps can generate BadAlloc, BadAtom, and BadWindow errors.

To obtain the XStandardColormap structure associated with the specified property, use XGetRGBColormaps.

Status XGetRGBColormaps(display, w, std_colormap_return, count_return, property)
    Display *display;
    Window w;
    XStandardColormap **std_colormap_return;
    int *count_return;
    Atom property;

display    Specifies the connection to the X server.
w         Specifies the window.
std_colormap_return      Returns the XStandardColormap structure.
count_return    Returns the number of colormaps.
property Specifies the property name.

The XGetRGBColormaps function returns the RGB colormap definitions stored in the specified property on the named window. If the property exists, is of type RGB_COLOR_MAP, is of format 32, and is long enough to contain a colormap definition, XGetRGBColormaps allocates and fills in space for the returned colormaps and returns a nonzero status. If the visualid is not present, XGetRGBColormaps assumes the default visual for the screen on which the window is located; if the killid is not present, None is assumed, which indicates that the resources cannot be released. Otherwise, none of the fields are set, and XGetRGBColormaps returns a zero status. Note that it is the caller's responsibility to honor the ICCCM restriction that only RGB_DEFAULT_MAP contain more than one definition.

XGetRGBColormaps can generate BadAtom and BadWindow errors.
Chapter 15

Resource Manager Functions

A program often needs a variety of options in the X environment (for example, fonts, colors, icons, and cursors). Specifying all of these options on the command line is awkward because users may want to customize many aspects of the program and need a convenient way to establish these customizations as the default setting. The resource manager is provided for this purpose. Resource specifications are usually stored in human-readable files and in server properties.

The resource manager is a database manager with a twist. In most database systems, you perform a query using an imprecise specification, and you get back a set of records. The resource manager, however, allows you to specify a large set of values with an imprecise specification, to query the database with a precise specification, and to get back only a single value. This should be used by applications that need to know what the user prefers for colors, fonts, and other resources. It is this use as a database for dealing with X resources that inspired the name "Resource Manager," although the resource manager can be and is used in other ways.

For example, a user of your application may want to specify that all windows should have a blue background but that all mail-reading windows should have a red background. With well-engineered and coordinated applications, a user can define this information using only two lines of specifications.

As an example of how the resource manager works, consider a mail-reading application called xmh. Assume that it is designed so that it uses a complex window hierarchy all the way down to individual command buttons, which may be actual small subwindows in some toolkits. These are often called objects or widgets. In such toolkit systems, each user interface object can be composed of other objects and can be assigned a name and a class. Fully qualified names or classes can have arbitrary numbers of component names, but a fully qualified name always has the same number of component names as a fully qualified class. This generally reflects the structure of the application as composed of these objects, starting with the application itself.

For example, the xmh mail program has a name "xmh" and is one of a class of "Mail" programs. By convention, the first character of class components is capitalized, and the first letter of name components is in lowercase. Each name and class finally has an attribute (for example "foreground" or "font"). If each window is properly assigned a name and class, it is easy for the user to specify attributes of any portion of the application.

At the top level, the application might consist of a paned window (that is, a window divided into several sections) named "toc". One pane of the paned window is a button box window named "buttons" and is filled with command buttons. One of these command buttons is used to incorporate new mail and has the name "incorporate". This window has a fully qualified name, "xmh.toc.buttons.incorporate", and a fully qualified class, "Xmh.Paned.Box.Command". Its fully qualified name is the name of its parent, "xmh.toc.buttons", followed by its name, "incorporate". Its class is the class of its parent, "Xmh.Paned.Box", followed by its particular class, "Command". The fully qualified name of a resource is the attribute's name appended to the object's fully qualified name, and the fully qualified class is the class appended to the object's class.

The incorporate button might need the following resources: Title string, Font, Foreground color for its inactive state, Background color for its inactive state, Foreground color for its active state, and Background color for its active state. Each resource is considered to be an attribute of the button and, as such, has a name and a class. For example, the foreground color for the
button in its active state might be named "activeForeground", and its class might be "Foreground".

When an application looks up a resource (for example, a color), it passes the complete name and complete class of the resource to a lookup routine. The resource manager compares this complete specification against the incomplete specifications of entries in the resource database, find the best match, and returns the corresponding value for that entry.

The definitions for the resource manager are contained in <X11/Xresource.h>.

15.1. Resource File Syntax

The syntax of a resource file is a sequence of resource lines terminated by newline characters or end of file. The syntax of an individual resource line is:

```
ResourceLine = Comment | IncludeFile | ResourceSpec | <empty line>
Comment = "!" {<any character except null or newline>}
IncludeFile = "#" WhiteSpace "include" WhiteSpace FileName WhiteSpace
FileName = <valid filename for operating system>
ResourceSpec = WhiteSpace ResourceName WhiteSpace ":" WhiteSpace Value
ResourceName = [Binding] [Component Binding] ComponentName
Binding = "." | "," | ":" | ":"
WhiteSpace = {<empty> | <horizontal tab>}
Component = "?" | ComponentName
ComponentName = NameChar {NameChar}
NameChar = "a"-"z" | "A"-"Z" | "0"-"9" | " _ " | "." | "*"
Value = {<any character except null or unescaped newline>}
```

Elements separated by vertical bar (|) are alternatives. Curly braces ({{...}}) indicate zero or more repetitions of the enclosed elements. Square brackets ([...]) indicate that the enclosed element is optional. Quotes ("...") are used around literal characters.

IncludeFile lines are interpreted by replacing the line with the contents of the specified file. The word "include" must be in lowercase. The filename is interpreted relative to the directory of the file in which the line occurs (for example, if the filename contains no directory or contains a relative directory specification).

If a ResourceName contains a contiguous sequence of two or more Binding characters, the sequence will be replaced with a single "." character if the sequence contains only "." characters, otherwise the sequence will be replaced with a single "*" character.

A resource database never contains more than one entry for a given ResourceName. If a resource file contains multiple lines with the same ResourceName, the last line in the file is used.

Any whitespace character before or after the name or colon in a ResourceSpec are ignored. To allow a Value to begin with whitespace, the two-character sequence "space" (backslash followed by space) is recognized and replaced by a space character, and the two-character sequence "tab" (backslash followed by horizontal tab) is recognized and replaced by a horizontal tab character. To allow a Value to contain embedded newline characters, the two-character sequence "n" is recognized and replaced by a newline character. To allow a Value to be broken across multiple lines in a text file, the two-character sequence "newline" (backslash followed by newline) is recognized and removed from the value. To allow a Value to contain arbitrary character codes, the four-character sequence "nnn", where each n is a digit character in the range of "0"-"7", is recognized and replaced with a single byte that contains the octal value specified by the sequence. Finally, the two-character sequence "\" is recognized and replaced with a single backslash.

As an example of these sequences, the following resource line contains a value consisting of four characters: a backslash, a null, a "z", and a newline:

```
magic.values: \000\n```
15.2. Resource Manager Matching Rules

The algorithm for determining which resource database entry matches a given query is the heart of the resource manager. All queries must fully specify the name and class of the desired resource (use of "*" and "?" are not permitted). The library supports up to 100 components in a full name or class. Resources are stored in the database with only partially specified names and classes, using pattern matching constructs. An asterisk (*) is a loose binding and is used to represent any number of intervening components, including none. A period (.) is a tight binding and is used to separate immediately adjacent components. A question mark (?) is used to match any single component name or class. A database entry cannot end in a loose binding; the final component (which cannot be "?") must be specified. The lookup algorithm searches the database for the entry that most closely matches (is most specific for) the full name and class being queried. When more than one database entry matches the full name and class, precedence rules are used to select just one.

The full name and class are scanned from left to right (from highest level in the hierarchy to lowest), one component at a time. At each level, the corresponding component and/or binding of each matching entry is determined, and these matching components and bindings are compared according to precedence rules. Each of the rules is applied at each level, before moving to the next level, until a rule selects a single entry over all others. The rules (in order of precedence) are:

1. An entry that contains a matching component (whether name, class, or "?") takes precedence over entries that elide the level (that is, entries that match the level in a loose binding).
2. An entry with a matching name takes precedence over both entries with a matching class and entries that match using "?". An entry with a matching class takes precedence over entries that match using "?".
3. An entry preceded by a tight binding takes precedence over entries preceded by a loose binding.

To illustrate these rules, consider the following resource database entries:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Binding</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmh<em>Paned</em>activeForeground:</td>
<td>red</td>
<td>(entry A)</td>
</tr>
<tr>
<td>*incorporate.Foreground:</td>
<td>blue</td>
<td>(entry B)</td>
</tr>
<tr>
<td>xmh.toc<em>Command</em>activeForeground:</td>
<td>green</td>
<td>(entry C)</td>
</tr>
<tr>
<td>xmh.toc()?.Foreground:</td>
<td>white</td>
<td>(entry D)</td>
</tr>
<tr>
<td>xmh.toc*Command.activeForeground:</td>
<td>black</td>
<td>(entry E)</td>
</tr>
</tbody>
</table>

Consider a query for the resource:

```
Xmh.toc.messagefunctions.incorporate.activeForeground (name)
Xmh.Paned.Box.Command.Foreground (class)
```

At the first level (xmh, Xmh) rule 1 eliminates entry B. At the second level (toc, Paned) rule 2 eliminates entry A. At the third level (messagefunctions, Box) no entries are eliminated. At the fourth level (incorporate, Command) rule 2 eliminates entry D. At the fifth level (activeForeground, Foreground) rule 3 eliminates entry C.

15.3. Quarks

Most uses of the resource manager involve defining names, classes, and representation types as string constants. However, always referring to strings in the resource manager can be slow, because it is so heavily used in some toolkits. To solve this problem, a shorthand for a string is used in place of the string in many of the resource manager functions. Simple comparisons can be performed rather than string comparisons. The shorthand name for a string is called a...
quark and is the type \texttt{XrmQuark}. On some occasions, you may want to allocate a quark that has no string equivalent. 
A quark is to a string what an atom is to a string in the server, but its use is entirely local to your application.

To allocate a new quark, use \texttt{XrmUniqueQuark}.
\begin{verbatim}
XrmQuark XrmUniqueQuark()
\end{verbatim}
The \texttt{XrmUniqueQuark} function allocates a quark that is guaranteed not to represent any string that is known to the resource manager.

Each name, class, and representation type is typedef'd as an \texttt{XrmQuark}.
\begin{verbatim}
typedef int XrmQuark, *XrmQuarkList;
typedef XrmQuark XrmName;
typedef XrmQuark XrmClass;
typedef XrmQuark XrmRepresentation;
#define NULLQUARK ((XrmQuark) 0)
\end{verbatim}
Lists are represented as null-terminated arrays of quarks. The size of the array must be large enough for the number of components used.
\begin{verbatim}
typedef XrmQuarkList XrmNameList;
typedef XrmQuarkList XrmClassList;
\end{verbatim}

To convert a string to a quark, use \texttt{XrmStringToQuark} or \texttt{XrmPermStringToQuark}.
\begin{verbatim}
#define XrmStringToName(string) XrmStringToQuark(string)
#define XrmStringToClass(string) XrmStringToQuark(string)
#define XrmStringToRepresentation(string) XrmStringToQuark(string)
XrmQuark XrmStringToQuark(string)
\end{verbatim}
\begin{verbatim}
char *string;
\end{verbatim}
\begin{verbatim}
XrmQuark XrmPermStringToQuark(string)
\end{verbatim}
\begin{verbatim}
char *string;
\end{verbatim}
\begin{verbatim}
string Specifies the string for which a quark is to be allocated.
\end{verbatim}
These functions can be used to convert from string to quark representation. If the string is not in the Host Portable Character Encoding the conversion is implementation dependent. The string argument to \texttt{XrmStringToQuark} need not be permanently allocated storage. \texttt{XrmPermStringToQuark} is just like \texttt{XrmStringToQuark}, except that Xlib is permitted to assume the string argument is permanently allocated, and hence that it can be used as the value to be returned by \texttt{XrmQuarkToString}.

To convert a quark to a string, use \texttt{XrmQuarkToString}.
\begin{verbatim}
#define XrmNameToString(name) XrmQuarkToString(name)
#define XrmClassToString(class) XrmQuarkToString(class)
#define XrmRepresentationToString(type) XrmQuarkToString(type)
char *XrmQuarkToString(quark)
\end{verbatim}
\begin{verbatim}
XrmQuark quark;
\end{verbatim}
\begin{verbatim}
quark Specifies the quark for which the equivalent string is desired.
\end{verbatim}
This function can be used to convert from quark representation to string. The string pointed to by the return value must not be modified or freed. The returned string is byte-for-byte equal to the original string passed to one of the string-to-quark routines. If no string exists for that quark, XrmQuarkToString returns NULL. For any given quark, if XrmQuarkToString returns a non-NULL value, all future calls will return the same value (identical address).

To convert a string with one or more components to a quark list, use XrmStringToQuarkList.

```c
#define XrmStringToNameList(str, name) XrmStringToQuarkList((str), (name))
#define XrmStringToClassList(str, class) XrmStringToQuarkList((str), (class))
```

```c
void XrmStringToQuarkList(string, quarks_return)
    char *string;
    XrmQuarkList quarks_return;

    string Specifies the string for which a quark list is to be allocated.
    quarks_return Returns the list of quarks.
```

The XrmStringToQuarkList function converts the null-terminated string (generally a fully qualified name) to a list of quarks. Note that the string must be in the valid ResourceName format (see section 15.1). If the string is not in the Host Portable Character Encoding the conversion is implementation dependent.

A binding list is a list of type XrmBindingList and indicates if components of name or class lists are bound tightly or loosely (that is, if wildcarding of intermediate components is specified).

```c
typedef enum {XrmBindTightly, XrmBindLoosely} XrmBinding, *XrmBindingList;
```

XrmBindTightly indicates that a period separates the components, and XrmBindLoosely indicates that an asterisk separates the components.

To convert a string with one or more components to a binding list and a quark list, use XrmStringToBindingQuarkList.

```c
void XrmStringToBindingQuarkList(string, bindings_return, quarks_return)
    char *string;
    XrmBindingList bindings_return;
    XrmQuarkList quarks_return;
```

```c
    string Specifies the string for which a quark list is to be allocated.
    bindings_return Returns the binding list. The caller must allocate sufficient space for the binding list before calling XrmStringToBindingQuarkList.
    quarks_return Returns the list of quarks. The caller must allocate sufficient space for the quarks list before calling XrmStringToBindingQuarkList.
```

Component names in the list are separated by a period or an asterisk character. The string must be in the format of a valid ResourceName (see section 15.1). If the string does not start with a period or an asterisk, a tight binding is assumed. For example, "**a.b*c" becomes:

```c
quarks:     a      b      c
bindings:    loose  tight  loose
```

15.4. Creating and Storing Databases

A resource database is an opaque type, XrmDatabase. Each database value is stored in an XrmValue structure. This structure consists of a size, an address, and a representation type. The size is specified in bytes. The representation type is a way for you to store data tagged by
some application-defined type (for example, "font" or "color"). It has nothing to do with the
C data type or with its class. The XrmValue structure is defined as:

typedef struct {
    unsigned int size;
    XPointer addr,
} XrmValue, *XrmValuePtr;

* To initialize the resource manager, use XrmInitialize.
void XrmInitialize();

To retrieve a database from disk, use XrmGetFileDatabase.
XrmDatabase XrmGetFileDatabase(filename)
    char *filename;

filename Specifies the resource database file name.
The XrmGetFileDatabase function opens the specified file, creates a new resource database,
and loads it with the specifications read in from the specified file. The specified file must con¬
tain a sequence of entries in valid ResourceLine format (see section 15.1). The file is parsed
in the current locale, and the database is created in the current locale. If it cannot open the
specified file, XrmGetFileDatabase returns NULL.

To store a copy of a database to disk, use XrmPutFileDatabase.
void XrmPutFileDatabase(database, stored_db)
    XrmDatabase database;
    char *stored_db;

database Specifies the database that is to be used.
stored_db Specifies the file name for the stored database.
The XrmPutFileDatabase function stores a copy of the specified database in the specified file.
Text is written to the file as a sequence of entries in valid ResourceLine format (see section
15.1). The file is written in the locale of the database. Entries containing resource names that
are not in the Host Portable Character Encoding, or containing values that are not in the encod¬
ing of the database locale, are written in an implementation-dependent manner. The order in
which entries are written is implementation dependent. Entries with representation types other
than "String" are ignored.

To obtain a pointer to the screen-independent resources of a display, use XResourceManager-
String.
char *XResourceManagerString(display)
    Display *display;

display Specifies the connection to the X server.
The XResourceManagerString returns the RESOURCE_MANAGER property from the
server's root window of screen zero, which was returned when the connection was opened
using XOpenDisplay. The property is converted from type STRING to the current locale.
The conversion is identical to that produced by XmbTextPropertyToTextList for a singleton
STRING property. The returned string is owned by Xlib, and should not be freed by the
client. Note that the property value must be in a format that is acceptable to XrmGetString-
Database. If no property exists, NULL is returned.

To obtain a pointer to the screen-specific resources of a screen, use XScreenResourceString.
char *XScreenResourceString(screen)
    Screen *screen;

screen          Specifies the screen.
The XStringResourceString returns the SCREEN_RESOURCES property from the root window of the specified screen. The property is converted from type STRING to the current locale. The conversion is identical to that produced by XmbTextPropertyToTextList for a singleton STRING property. Note that the property value must be in a format that is acceptable to XrmGetStringDatabase. If no property exists, NULL is returned. The caller is responsible for freeing the returned string, using XFree.

To create a database from a string, use XrmGetStringDatabase.

XrmDatabase XrmGetStringDatabase(data)
    char *data;

data          Specifies the database contents using a string.
The XrmGetStringDatabase function creates a new database and stores the resources specified in the specified null-terminated string. XrmGetStringDatabase is similar to XrmGetFileDatabase except that it reads the information out of a string instead of out of a file. The string must contain a sequence of entries in valid ResourceLine format (see section 15.1). The string is parsed in the current locale, and the database is created in the current locale.

To obtain locale name of a database, use XrmLocaleOfDatabase.

cchar *XrmLocaleOfDatabase(database)
    XrmDatabase database;

database       Specifies the resource database.
The XrmLocaleOfDatabase function returns the name of the locale bound to the specified database, as a null-terminated string. The returned locale name string is owned by Xlib and should not be modified or freed by the client. Xlib is not permitted to free the string until the database is destroyed. Until the string is freed, it will not be modified by Xlib.

To destroy a resource database and free its allocated memory, use XrmDestroyDatabase.

void XrmDestroyDatabase(database)
    XrmDatabase database;

database       Specifies the resource database.
If database is NULL, XrmDestroyDatabase returns immediately.

To associate a resource database with a display, use XrmSetDatabase.

void XrmSetDatabase(display, database)
    Display *display;
        XrmDatabase database;

display        Specifies the connection to the X server.
database       Specifies the resource database.
The XrmSetDatabase function associates the specified resource database (or NULL) with the specified display. The database previously associated with the display (if any) is not destroyed. A client or toolkit may find this function convenient for retaining a database once it is constructed.
To get the resource database associated with a display, use `XrmGetDatabase`.

```c
XrmDatabase XrmGetDatabase(display)
    Display *display;
```

display Specifies the connection to the X server.

The `XrmGetDatabase` function returns the database associated with the specified display. It returns NULL if a database has not yet been set.

15.5. Merging Resource Databases

To merge the contents of a resource file into a database, use `XrmCombineFileDatabase`.

```c
void XrmCombineFileDatabase(filename, target_db, override)
    char *filename;
    XrmDatabase *target_db;
    Bool override;
```

filename Specifies the resource database file name.

target_db Specifies the resource database into which the source database is to be merged.

The `XrmCombineFileDatabase` function merges the contents of a resource file into a database. If the same specifier is used for an entry in both the file and the database, the entry in the file will replace the entry in the database if `override` is True; otherwise, the entry in file is discarded. The file is parsed in the current locale. If the file cannot be read a zero status is returned; otherwise a nonzero status is returned. If `target_db` contains NULL, `XrmCombineFileDatabase` creates and returns a new database to it. Otherwise, the database pointed to by `target_db` is not destroyed by the merge. The database entries are merged without changing values or types, regardless of the locale of the database. The locale of the target database is not modified.

To merge the contents of one database into another database, use `XrmCombineDatabase`.

```c
void XrmCombineDatabase(source_db, target_db, override)
    XrmDatabase source_db, *target_db;
    Bool override;
```

source_db Specifies the resource database that is to be merged into the target database.

target_db Specifies the resource database into which the source database is to be merged.

override Specifies whether source entries override target ones.

The `XrmCombineDatabase` function merges the contents of one database into another. If the same specifier is used for an entry in both databases, the entry in the source_db will replace the entry in the target_db if `override` is True; otherwise, the entry in source_db is discarded. If `target_db` contains NULL, `XrmCombineDatabase` simply stores source_db in it. Otherwise, source_db is destroyed by the merge, but the database pointed to by `target_db` is not destroyed. The database entries are merged without changing values or types, regardless of the locales of the databases. The locale of the target database is not modified.

To merge the contents of one database into another database with override semantics, use `XrmMergeDatabases`.

```c
void XrmMergeDatabases(source_db, target_db)
    XrmDatabase source_db, *target_db;
```

source_db Specifies the resource database that is to be merged into the target database.

target_db Specifies the resource database into which the source database is to be merged.

The `XrmMergeDatabases` function merges the contents of one database into another. If the same specifier is used for an entry in both databases, the entry in the source_db will replace
the entry in the target_db (that is, it overrides target_db). If target_db contains NULL, XrmMergeDatabases simply stores source_db in it. Otherwise, source_db is destroyed by the merge, but the database pointed to by target_db is not destroyed. The database entries are merged without changing values or types, regardless of the locales of the databases. The locale of the target database is not modified.

15.6. Looking Up Resources
To retrieve a resource from a resource database, use XrmGetResource, XrmQGetResource, or XrmQGetSearchResource.

**Bool XrmGetResource(database, str_name, str_class, str_type_return, value_return)**

XrmDatabase database;
char *str_name;
char *str_class;
char **str_type_return;
XrmValue *value_return;

-database Specifies the database that is to be used.
-str_name Specifies the fully qualified name of the value being retrieved (as a string).
-str_class Specifies the fully qualified class of the value being retrieved (as a string).
-str_type_return Returns the representation type of the destination (as a string).
-value_return Returns the value in the database.

**Bool XrmQGetResource(database, quark_name, quark_class, quark_type_return, value_return)**

XrmDatabase database;
XrmNameList quark_name;
XrmClassList quark_class;
XrmRepresentation *quark_type_return;
XrmValue *value_return;

-database Specifies the database that is to be used.
-quark_name Specifies the fully qualified name of the value being retrieved (as a quark).
-quark_class Specifies the fully qualified class of the value being retrieved (as a quark).
-quark_type_return Returns the representation type of the destination (as a quark).
-value_return Returns the value in the database.

The XrmGetResource and XrmQGetResource functions retrieve a resource from the specified database. Both take a fully qualified name/class pair, a destination resource representation, and the address of a value (size/address pair). The value and returned type point into database memory; therefore, you must not modify the data.

The database only frees or overwrites entries on XrmPutResource, XrmQPutResource, or XrmMergeDatabases. A client that is not storing new values into the database or is not merging the database should be safe using the address passed back at any time until it exits. If a resource was found, both XrmGetResource and XrmQGetResource return True; otherwise, they return False.

Most applications and toolkits do not make random probes into a resource database to fetch resources. The X toolkit access pattern for a resource database is quite stylized. A series of from 1 to 20 probes are made with only the last name/class differing in each probe. The XrmGetResource function is at worst a $2^n$ algorithm, where $n$ is the length of the name/class list. This can be improved upon by the application programmer by prefetching a list of

297
database levels that might match the first part of a name/class list.

To obtain a list of database levels, use `XrmQGetSearchList`.

typedef XrmHashTable *XrmSearchList;

`Bool XrmQGetSearchList(database, names, classes, list_return, list_length)`

- `database`: Specifies the database that is to be used.
- `names`: Specifies a list of resource names.
- `classes`: Specifies a list of resource classes.
- `list_return`: Returns a search list for further use. The caller must allocate sufficient space for the list before calling `XrmQGetSearchList`.
- `list_length`: Specifies the number of entries (not the byte size) allocated for `list_return`.

The `XrmQGetSearchList` function takes a list of names and classes and returns a list of database levels where a match might occur. The returned list is in best-to-worst order and uses the same algorithm as `XrmGetResource` for determining precedence. If `list_return` was large enough for the search list, `XrmQGetSearchList` returns `True`; otherwise, it returns `False`.

The size of the search list that the caller must allocate is dependent upon the number of levels and wildcards in the resource specifiers that are stored in the database. The worst case length is $3^n$, where $n$ is the number of name or class components in `names` or `classes`.

When using `XrmQGetSearchList` followed by multiple probes for resources with a common name and class prefix, only the common prefix should be specified in the name and class list to `XrmQGetSearchList`.

To search resource database levels for a given resource, use `XrmQGetSearchResource`.

`Bool XrmQGetSearchResource(list, name, class, type_return, value_return)`

- `list`: Specifies the search list returned by `XrmQGetSearchList`.
- `name`: Specifies the resource name.
- `class`: Specifies the resource class.
- `type_return`: Returns data representation type.
- `value_return`: Returns the value in the database.

The `XrmQGetSearchResource` function searches the specified database levels for the resource that is fully identified by the specified name and class. The search stops with the first match. `XrmQGetSearchResource` returns `True` if the resource was found; otherwise, it returns `False`.

A call to `XrmQGetSearchList` with a name and class list containing all but the last component of a resource name followed by a call to `XrmQGetSearchResource` with the last component name and class returns the same database entry as `XrmGetResource` and `XrmQGetResource` with the fully qualified name and class.
15.7. Storing Into a Resource Database

To store resources into the database, use `XrmPutResource` or `XrmQPutResource`. Both functions take a partial resource specification, a representation type, and a value. This value is copied into the specified database.

```c
void XrmPutResource(database, specifier, type, value)
    XrmDatabase *database;
    char *specifier;
    char *type;
    XrmValue *value;

database Specifies the resource database.
specifier Specifies a complete or partial specification of the resource.
type Specifies the type of the resource.
value Specifies the value of the resource, which is specified as a string.
```

If database contains NULL, `XrmPutResource` creates a new database and returns a pointer to it. `XrmPutResource` is a convenience function that calls `XrmStringToBindingQuarkList` followed by:

```c
    XrmQPutResource(database, bindings, quarks, XrmStringToQuark(type), value)
```

If the specifier and type are not in the Host Portable Character Encoding the result is implementation dependent. The value is stored in the database without modification.

```c
void XrmQPutResource(database, bindings, quarks, type, value)
    XrmDatabase *database;
    XrmBindingList bindings;
    XrmQuarkList quarks;
    XrmRepresentation type;
    XrmValue *value;

database Specifies the resource database.
bindings Specifies a list of bindings.
quarks Specifies the complete or partial name or the class list of the resource.
type Specifies the type of the resource.
value Specifies the value of the resource, which is specified as a string.
```

If database contains NULL, `XrmQPutResource` creates a new database and returns a pointer to it. If a resource entry with the identical bindings and quarks already exists in the database, the previous value is replaced by the new specified value. The value is stored in the database without modification.

To add a resource that is specified as a string, use `XrmPutStringResource`.

```c
void XrmPutStringResource(database, specifier, value)
    XrmDatabase *database;
    char *specifier;
    char *value;

database Specifies the resource database.
specifier Specifies a complete or partial specification of the resource.
value Specifies the value of the resource, which is specified as a string.
```
If database contains NULL, \texttt{XrmPutStringResource} creates a new database and returns a pointer to it. \texttt{XrmPutStringResource} adds a resource with the specified value to the specified database. \texttt{XrmPutStringResource} is a convenience function that first calls \texttt{XrmStringToBindingQuarkList} on the specifier and then calls \texttt{XrmQPutResource}, using a "String" representation type. If the specifier is not in the Host Portable Character Encoding the result is implementation dependent. The value is stored in the database without modification.

To add a string resource using quarks as a specification, use \texttt{XrmQPutStringResource}.

```c
void XrmQPutStringResource(XrmDatabase *database, XrmBindingList bindings, XrmQuarkList quarks, char *value)
```

database\quad Specifies the resource database.
bindings\quad Specifies a list of bindings.
quarks\quad Specifies the complete or partial name or the class list of the resource.
value\quad Specifies the value of the resource, which is specified as a string.

If database contains NULL, \texttt{XrmQPutStringResource} creates a new database and returns a pointer to it. \texttt{XrmQPutStringResource} is a convenience routine that constructs an \texttt{XrmValue} for the value string (by calling \texttt{strlen} to compute the size) and then calls \texttt{XrmQPutResource}, using a "String" representation type. The value is stored in the database without modification.

To add a single resource entry that is specified as a string that contains both a name and a value, use \texttt{XrmPutLineResource}.

```c
void XrmPutLineResource(XrmDatabase *database, char *line)
```

database\quad Specifies the resource database.
line\quad Specifies the resource name and value pair as a single string.

If database contains NULL, \texttt{XrmPutLineResource} creates a new database and returns a pointer to it. \texttt{XrmPutLineResource} adds a single resource entry to the specified database. The line must be in valid ResourceLine format (see section 15.1). The string is parsed in the locale of the database. If the \texttt{ResourceName} is not in the Host Portable Character Encoding the result is implementation dependent. Note that comment lines are not stored.

### 15.8. Enumerating Database Entries

To enumerate the entries of a database, use \texttt{XrmEnumerateDatabase}.

```c
#define XrmEnumAllLevels \texttt{0}
#define XrmEnumOneLevel \texttt{1}

Bool XrmEnumerateDatabase(XrmDatabase database, XrmNameList name_prefix, XrmClassList class_prefix, int mode, Bool (*proc)(); XPointer arg)
```

database\quad Specifies the resource database.
name_prefix\quad Specifies the complete or partial name or the class list of the resource.
class_prefix\quad Specifies the complete or partial name or the class list of the resource.
mode\quad Specifies the mode of enumeration (0 for all, 1 for one level).
proc\quad Specifies the callback function to be called for each resource.
arg\quad Specifies the argument to be passed to the callback function.

If database contains NULL, \texttt{XrmEnumerateDatabase} creates a new database and returns a pointer to it. \texttt{XrmEnumerateDatabase} adds a single resource entry to the specified database. The line must be in valid ResourceLine format (see section 15.1). The string is parsed in the locale of the database. If the \texttt{ResourceName} is not in the Host Portable Character Encoding the result is implementation dependent. Note that comment lines are not stored.
The `XrmEnumerateDatabase` function calls the specified procedure for each resource in the database that would match some completion of the given name/class resource prefix. The order in which resources are found is implementation-dependent. If mode is `XrmEnumOneLevel`, then a resource must match the given name/class prefix with just a single name and class appended. If mode is `XrmEnumAllLevels`, the resource must match the given name/class prefix with one or more names and classes appended. If the procedure returns `True`, the enumeration terminates and the function returns `True`. If the procedure always returns `False`, all matching resources are enumerated and the function returns `False`.

The procedure is called with the following arguments:

\[
(*\text{proc})(\text{database}, \text{bindings}, \text{quarks}, \text{type}, \text{value}, \text{arg})
\]

\[
\text{XrmDatabase} * \text{database},
\text{XrmBindingList} \text{bindings},
\text{XrmQuarkList} \text{quarks},
\text{XrmRepresentation} * \text{type},
\text{XrmValue} * \text{value},
\text{XPointer} \text{closure};
\]

The bindings and quarks lists are terminated by NULLQUARK. Note that pointers to the database and type are passed, but these values should not be modified.

### 15.9 Parsing Command Line Options

The `XrmParseCommand` function can be used to parse the command line arguments to a program and modify a resource database with selected entries from the command line.

```c
typedef enum {  
    XrmoptionNoArg,  /* Value is specified in XrmOptionDescRec.value */  
    XrmoptionIsArg,  /* Value is the option string itself */  
    XrmoptionStickyArg,  /* Value is characters immediately following option */  
    XrmoptionSepArg,  /* Value is next argument in argv */  
    XrmoptionResArg,  /* Resource and value in next argument in argv */  
    XrmoptionSkipArg,  /* Ignore this option and the next argument in argv */  
    XrmoptionSkipLine,  /* Ignore this option and the rest of argv */  
    XrmoptionSkipNArgs  /* Ignore this option and the next XrmOptionDescRec.value arguments in argv */
} XrmOptionKind;
```

Note that `XrmoptionSkipArg` is equivalent to `XrmoptionSkipNArgs` with the `XrmOptionDescRec.value` field containing the value one. Note also that the value zero for `XrmoptionSkipNArgs` indicates that only the option itself is to be skipped.

```c
typedef struct {  
    char *option;  /* Option specification string in argv */  
    char *specifier;  /* Binding and resource name (sans application name) */  
    XrmOptionKind argKind;  /* Which style of option it is */  
    XPointer value;  /* Value to provide if XrmoptionNoArg or XrmoptionSkipNArgs */
} XrmOptionDescRec, *XrmOptionDescList;
```
To load a resource database from a C command line, use `XrmParseCommand`.

```c
void XrmParseCommand(database, table, table_count, name, argc_in_out, argv_in_out)
    XrmDatabase *database;
    XrmOptionDescList table;
    int table_count;
    char *name;
    int *argc_in_out;
    char **argv_in_out;
```

- `database` specifies the resource database.
- `table` specifies the table of command line arguments to be parsed.
- `table_count` specifies the number of entries in the table.
- `name` specifies the application name.
- `argc_in_out` specifies the number of arguments and returns the number of remaining arguments.
- `argv_in_out` specifies the command line arguments and returns the remaining arguments.

The `XrmParseCommand` function parses an `(argc, argv)` pair according to the specified option table, loads recognized options into the specified database with type "String," and modifies the (argc, argv) pair to remove all recognized options. If `database` contains NULL, `XrmParseCommand` creates a new database and returns a pointer to it. Otherwise, entries are added to the database specified. If a database is created, it is created in the current locale.

The specified table is used to parse the command line. Recognized options in the table are removed from `argv`, and entries are added to the specified resource database. The table entries contain information on the option string, the option name, the style of option, and a value to provide if the option kind is `XrmoptionNoArg`. The option names are compared byte-for-byte to arguments in `argv`, independent of any locale. The resource values given in the table are stored in the resource database without modification. All resource database entries are created using a "String" representation type. The `argc` argument specifies the number of arguments in `argv` and is set on return to the remaining number of arguments that were not parsed. The `name` argument should be the name of your application for use in building the database entry. The `name` argument is prefixed to the `resourceName` in the option table before storing a database entry. No separating (binding) character is inserted, so the table must contain either a period (.) or an asterisk (*) as the first character in each `resourceName` entry. To specify a more completely qualified resource name, the `resourceName` entry can contain multiple components. If the `name` argument and the `resourceNames` are not in the Host Portable Character Encoding the result is implementation dependent.

The following provides a sample option table:

```c
static XrmOptionDescRec opTable[] = {
    "-background", "background", XrmoptionSepArg, (XPointer) NULL,
    "-bd", "borderColor", XrmoptionSepArg, (XPointer) NULL,
    "-bg", "background", XrmoptionSepArg, (XPointer) NULL,
    "-borderwidth", "TopLevelShell.borderWidth", XrmoptionSepArg, (XPointer) NULL,
    "-bordercolor", "borderColor", XrmoptionSepArg, (XPointer) NULL,
    "-bw", "TopLevelShell.borderWidth", XrmoptionSepArg, (XPointer) NULL,
    "-display", "display", XrmoptionSepArg, (XPointer) NULL,
    "-fg", "foreground", XrmoptionSepArg, (XPointer) NULL,
    "-fn", "font", XrmoptionSepArg, (XPointer) NULL,
    "-font", "font", XrmoptionSepArg, (XPointer) NULL,
    "-foreground", "foreground", XrmoptionSepArg, (XPointer) NULL,
    "-geometry", "TopLevelShell.geometry", XrmoptionSepArg, (XPointer) NULL,
    "-image", "TopLevelShell.iconic", XrmoptionSepArg, (XPointer) NULL,
    "-name", "name", XrmoptionSepArg, (XPointer) NULL,
    "-display", "display", XrmoptionSepArg, (XPointer) NULL,
    "-foreground", "foreground", XrmoptionSepArg, (XPointer) NULL,
    "-geometry", "TopLevelShell.geometry", XrmoptionSepArg, (XPointer) NULL,
    "-iconic", "TopLevelShell.iconic", XrmoptionSepArg, (XPointer) NULL,
    "-name", "name", XrmoptionSepArg, (XPointer) NULL,
};
```
In this table, if the `-background` (or `-bg`) option is used to set background colors, the stored resource specifier matches all resources of attribute background. If the `-borderwidth` option is used, the stored resource specifier applies only to border width attributes of class TopLevelShell (that is, outer-most windows, including pop-up windows). If the `-title` option is used to set a window name, only the topmost application windows receive the resource.

When parsing the command line, any unique unambiguous abbreviation for an option name in the table is considered a match for the option. Note that uppercase and lowercase matter.
The \texttt{XKeycodeToKeysym} function uses internal Xlib tables and returns the \texttt{KeySym} defined for the specified \texttt{KeyCode} and the element of the \texttt{KeyCode} vector. If no symbol is defined, \texttt{XKeycodeToKeysym} returns \texttt{NoSymbol}.

To obtain a key code for a key having a specific \texttt{KeySym}, use \texttt{XKeysymToKeycode}.

\begin{verbatim}
KeyCode XKeysymToKeycode(display, keysym)
    Display *display;
    KeySym keysym;
    display Specifies the connection to the X server.
    keysym Specifies the KeySym that is to be searched for.
If the specified KeySym is not defined for any KeyCode, \texttt{XKeysymToKeycode} returns zero.
\end{verbatim}

The mapping between key codes and KeySyms is cached internal to Xlib. When this information is changed at the server, an Xlib function must be called to refresh the cache. To refresh the stored modifier and keymap information, use \texttt{XRefreshKeyboardMapping}.

\begin{verbatim}
XRefreshKeyboardMapping(event_map)
    XMappingEvent *event_map;
    event_map Specifies the mapping event that is to be used.
\end{verbatim}

The \texttt{XRefreshKeyboardMapping} function refreshes the stored modifier and keymap information. You usually call this function when a \texttt{MappingNotify} event with a request member of \texttt{MappingKeyboard} or \texttt{MappingModifier} occurs. The result is to update Xlib's knowledge of the keyboard.

KeySyms have string names as well as numeric codes. To convert the name of the \texttt{KeySym} to the \texttt{KeySym} code, use \texttt{XStringToKeysym}.

\begin{verbatim}
KeySym XStringToKeysym(string)
    char *string;
    string Specifies the name of the KeySym that is to be converted.
\end{verbatim}

Standard KeySym names are obtained from \texttt{<X11/keysymdef.h>} by removing the \texttt{XK_} prefix from each name. KeySyms that are not part of the Xlib standard also may be obtained with this function. Note that the set of KeySyms that are available in this manner and the mechanisms by which Xlib obtains them is implementation dependent.

If the keysym name is not in the Host Portable Character Encoding the result is implementation dependent. If the specified string does not match a valid KeySym, \texttt{XStringToKeysym} returns \texttt{NoSymbol}.

To convert a KeySym code to the name of the KeySym, use \texttt{XKeysymToString}.

\begin{verbatim}
char *XKeysymToString(keysym)
    KeySym keysym;
    keysym Specifies the KeySym that is to be converted.
\end{verbatim}

The returned string is in a static area and must not be modified. The returned string is in the Host Portable Character Encoding. If the specified KeySym is not defined, \texttt{XKeysymToString} returns a NULL.

16.1.1. Keysym Classification Macros

You may want to test if a KeySym is, for example, on the keypad or on one of the function keys. You can use the KeySym macros to perform the following tests.
Chapter 16
Application Utility Functions

Once you have initialized the X system, you can use the Xlib utility functions to:
• Obtain and classify KeySyms
• Allocate permanent storage
• Parse window geometry strings
• Manipulate regions
• Use cut buffers
• Determine the appropriate visual
• Manipulate images
• Manipulate bitmaps
• Use the context manager

As a group, the functions discussed in this chapter provide the functionality that is frequently needed and that spans toolkits. Many of these functions do not generate actual protocol requests to the server.

16.1. Keyboard Utility Functions

This section discusses mapping between KeyCodes and KeySyms, names for KeySyms, and KeySym classification macros. The functions in this section operate on a cached copy of the server keyboard mapping. The first four KeySyms for each key code are modified according to the rules given in section 12.7. If you want the untransformed KeySyms defined for a key, you should only use the functions described in section 12.7.

To obtain a KeySym for the key code of an event, use XLookupKeysym.

KeySym XLookupKeysym(key_event, index)

- key_event: Specifies the KeyPress or KeyRelease event.
- index: Specifies the index into the KeySyms list for the event's KeyCode.

The XLookupKeysym function uses a given keyboard event and the index you specified to return the KeySym from the list that corresponds to the KeyCode member in the XKeyPressedEvent or XKeyReleasedEvent structure. If no KeySym is defined for the KeyCode of the event, XLookupKeysym returns NoSymbol.

To obtain a KeySym for a specific key code, use XKeycodeToKeysym.

KeySym XKeycodeToKeysym(display, key_code, index)

- display: Specifies the connection to the X server.
- key_code: Specifies the KeyCode.
- index: Specifies the element of KeyCode vector.
IsCursorKey(keysyr)

keysym Specifies the KeySym that is to be tested.
Returns True if the specified KeySym is a cursor key.

IsFunctionKey(keysyr)

keysym Specifies the KeySym that is to be tested.
Returns True if the specified KeySym is a function key.

IsKeypadKey(keysyr)

keysym Specifies the KeySym that is to be tested.
Returns True if the specified KeySym is a keypad key.

IsMiscFunctionKey(keysyr)

keysym Specifies the KeySym that is to be tested.
Returns True if the specified KeySym is a miscellaneous function key.

IsModifierKey(keysyr)

keysym Specifies the KeySym that is to be tested.
Returns True if the specified KeySym is a modifier key.

IsPFKey(keysyr)

keysym Specifies the KeySym that is to be tested.
Returns True if the specified KeySym is a PF key.

16.2. Latin-1 Keyboard Event Functions

Chapter 13 describes internationalized text input facilities, but sometimes it is expedient to write an application that only deals with Latin-1 characters and ASCII controls, so Xlib provides a simple function for that purpose. XLookupString handles the standard modifier semantics described in section 12.7. This function does not use any of the input method facilities described in chapter 13, and does not depend on the current locale.

To map a key event to an ISO Latin-1 string, use XLookupString.

```
int XLookupString(event_struct, buffer_return, bytes_buffer, keysym_return, status_in_out)

XKeyEvent *event_struct;
char *buffer_return;
int bytes_buffer;
KeySym *keysym_return;
XComposeStatus *status_in_out;

event_struct Specifies the key event structure to be used. You can pass XKeyPressedEvent or XKeyReleasedEvent.
buffer_return Returns the translated characters.
bytes_buffer Specifies the length of the buffer. No more than bytes_buffer of translation are returned.
```
**keysym_return** Returns the KeySym computed from the event if this argument is not NULL.

**status_in_out** Specifies or returns the XComposeStatus structure or NULL.

The XLookupString function translates a key event to a KeySym and a string. The KeySym is obtained by using the standard interpretation of the Shift, Lock, and group modifiers as defined in the X Protocol specification. If the KeySym has been rebound (see XRebindKeysym), the bound string will be stored in the buffer. Otherwise, the KeySym is mapped, if possible, to an ISO Latin-1 character or (if the Control modifier is on) to an ASCII control character, and that character is stored in the buffer. XLookupString returns the number of characters that are stored in the buffer.

If present (non-NULL), the XComposeStatus structure records the state, which is private to Xlib, that needs preservation across calls to XLookupString to implement compose processing. The creation of XComposeStatus structures is implementation dependent; a portable program must pass NULL for this argument.

XLookupString depends on the cached keyboard information mentioned in the previous section, so it is necessary to use XRefreshKeyboardMapping to keep this information up to date.

To rebind the meaning of a KeySym for XLookupString, use XRebindKeysym.

XRebindKeysym(display, keysym, list, mod_count, string, num_bytes)

- **display** Specifies the connection to the X server.
- **keysym** Specifies the KeySym that is to be rebound.
- **list** Specifies the KeySyms to be used as modifiers.
- **mod_count** Specifies the number of modifiers in the modifier list.
- **string** Specifies the string that is copied and will be returned by XLookupString.
- **num_bytes** Specifies the number of bytes in the string argument.

The XRebindKeysym function can be used to rebind the meaning of a KeySym for the client. It does not redefine any key in the X server but merely provides an easy way for long strings to be attached to keys. XLookupString returns this string when the appropriate set of modifier keys are pressed and when the KeySym would have been used for the translation. No text conversions are performed; the client is responsible for supplying appropriately encoded strings. Note that you can rebind a KeySym that may not exist.

### 16.3. Allocating Permanent Storage

To allocate some memory you will never give back, use Xpermalloc.

```c
char *Xpermalloc(size)

unsigned int size;
```

The Xpermalloc function allocates storage that can never be freed for the life of the program. The memory is allocated with alignment for the C type double. This function may provide some performance and space savings over the standard operating system memory allocator.
16.4. Parsing the Window Geometry

To parse standard window geometry strings, use XParseGeometry.

```c
int XParseGeometry(char *parsestring, int *x_return, int *y_return, unsigned int *width_return, unsigned int *height_return);
```

- `parsestring` Specifies the string you want to parse.
- `x_return` Return the x and y offsets.
- `y_return` Return the x and y offsets.
- `width_return` Return the width and height determined.
- `height_return` Return the width and height determined.

By convention, X applications use a standard string to indicate window size and placement. XParseGeometry makes it easier to conform to this standard because it allows you to parse the standard window geometry. Specifically, this function lets you parse strings of the form:

```
[=]|<width>|<X><height>]|{+-}<xoffset>|{+-}<yoffset>]
```

The fields map into the arguments associated with this function. (Items enclosed in <> are integers, items in [] are optional, and items enclosed in {} indicate “choose one of.”) Note that the brackets should not appear in the actual string.) If the string is not in the Host Portable Character Encoding the result is implementation dependent.

The XParseGeometry function returns a bitmask that indicates which of the four values (width, height, xoffset, and yoffset) were actually found in the string and whether the x and y values are negative. By convention, -0 is not equal to +0, because the user needs to be able to say “position the window relative to the right or bottom edge.” For each value found, the corresponding argument is updated. For each value not found, the argument is left unchanged. The bits are represented by XValue, YValue, WidthValue, HeightValue, XNegative, or YNegative and are defined in <X11/Xutil.h>. They will be set whenever one of the values is defined or one of the signs is set.

If the function returns either the XValue or YValue flag, you should place the window at the requested position.

To construct a window’s geometry information, use XWMGeometry.

```c
int XWMGeometry(Display *display, Screen screen, char *user_geom, char *def_geom, unsigned int bwidth, XSizeHints *hints, int *x_return, int *y_return, int *width_return, int *height_return, int *gravity_return);
```

- `display` Specifies the connection to the X server.
- `screen` Specifies the screen.
- `user_geom` Specifies the user-specified geometry or NULL.
- `def_geom` Specifies the application’s default geometry or NULL.
`bwidth` Specifies the border width.

`hints` Specifies the size hints for the window in its normal state.

`x_return` Return the x and y offsets.

`y_return` Return the width and height determined.

`width_return` Return the width and height determined.

`height_return` Returns the window gravity.

The `XWMGeometry` function combines any geometry information (given in the format used by `XParseGeometry`) specified by the user and by the calling program with size hints (usually the ones to be stored in `WM_NORMAL_HINTS`) and returns the position, size, and gravity (`NorthWestGravity`, `NorthEastGravity`, `SouthEastGravity`, or `SouthWestGravity`) that describe the window. If the base size is not set in the `XSizeHints` structure, the minimum size is used if set. Otherwise, a base size of zero is assumed. If no minimum size is set in the hints structure, the base size is used. A mask (in the form returned by `XParseGeometry`) that describes which values came from the user specification and whether or not the position coordinates are relative to the right and bottom edges is returned. Note that these coordinates will have already been accounted for in the `x_return` and `y_return` values.

Note that invalid geometry specifications can cause a width or height of zero to be returned. The caller may pass the address of the hints `win_gravity` field as `gravity_return` to update the hints directly.

16.5. Manipulating Regions

Regions are arbitrary sets of pixel locations. Xlib provides functions for manipulating regions. The opaque type `Region` is defined in `<X11/Xutil.h>`. Xlib provides functions that you can use to manipulate regions. This section discusses how to:

- Create, copy, or destroy regions
- Move or shrink regions
- Compute with regions
- Determine if regions are empty or equal
- Locate a point or rectangle in a region

16.5.1. Creating, Copying, or Destroying Regions

To create a new empty region, use `XCreateRegion`.

```
Region XCreateRegion()
```

To generate a region from a polygon, use `XPolygonRegion`.

```
Region XPolygonRegion(points, n, fill_rule)
    XPoint points[];
    int n;
    int fill_rule;

    points Specifies an array of points.
    n Specifies the number of points in the polygon.
    fill_rule Specifies the fill-rule you want to set for the specified GC. You can pass
                 `EvenOddRule` or `WindingRule`.
```

The `XPolygonRegion` function returns a region for the polygon defined by the points array. For an explanation of `fill_rule`, see `XCreateGC`.

To set the clip-mask of a GC to a region, use `XSetRegion`.
XSetRegion(display, gc, r)
    Display *display;
    GC gc;
    Region r;

display Specifies the connection to the X server.
gc Specifies the GC.
r Specifies the region.

The XSetRegion function sets the clip-mask in the GC to the specified region. Once it is set in the GC, the region can be destroyed.

To deallocate the storage associated with a specified region, use XDestroyRegion.
XDestroyRegion(r)
    Region r;

16.5.2. Moving or Shrinking Regions
To move a region by a specified amount, use XOffsetRegion.
XOffsetRegion(r, dx, dy)
    Region r;
    int dx, dy;

r Specifies the region.
dx dy Specify the x and y coordinates, which define the amount you want to move the specified region.

To reduce a region by a specified amount, use XShrinkRegion.
XShrinkRegion(r, dx, dy)
    Region r;
    int dx, dy;

r Specifies the region.
dx dy Specify the x and y coordinates, which define the amount you want to shrink the specified region.

Positive values shrink the size of the region, and negative values expand the region.

16.5.3. Computing with Regions
To generate the smallest rectangle enclosing a region, use XClipBox.
XClipBox(r, rect_return)
    Region r;
    XRectangle *rect_return;

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

The XClipBox function returns the smallest rectangle enclosing the specified region.

To compute the intersection of two regions, use XIntersectRegion.
XIntersectRegion(sra, srb, dr_return)

Region sra, srb, dr_return;

sra
srb
Specify the two regions with which you want to perform the computation.
dr_return
Returns the result of the computation.

To compute the union of two regions, use XUnionRegion.

XUnionRegion(sra, srb, dr_return)

Region sra, srb, dr_return;

sra
srb
Specify the two regions with which you want to perform the computation.
dr_return
Returns the result of the computation.

To create a union of a source region and a rectangle, use XUnionRectWithRegion.

XUnionRectWithRegion(rectangle, src_region, dest_region_return)

XRectangle *rectangle;
Region src_region;
Region dest_region_return;

rectangle
Specifies the rectangle.
src_region
Specifies the source region to be used.
dest_region_return
Returns the destination region.

The XUnionRectWithRegion function updates the destination region from a union of the specified rectangle and the specified source region.

To subtract two regions, use XSubtractRegion.

XSubtractRegion(sra, srb, dr_return)

Region sra, srb, dr_return;

sra
srb
Specify the two regions with which you want to perform the computation.
dr_return
Returns the result of the computation.

The XSubtractRegion function subtracts srb from sra and stores the results in dr_return.

To calculate the difference between the union and intersection of two regions, use XXorRegion.

XXorRegion(sra, srb, dr_return)

Region sra, srb, dr_return;

sra
srb
Specify the two regions with which you want to perform the computation.
dr_return
Returns the result of the computation.

16.5.4. Determining if Regions Are Empty or Equal

To determine if the specified region is empty, use XEmptyRegion.

Bool XEmptyRegion(r)

Region r;
'r' Specifies the region.
The XEmptyRegion function returns True if the region is empty.

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

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

r1 r2 Specify the two regions.
The XEqualRegion function returns True if the two regions have the same offset, size, and shape.

16.5.5. Locating a Point or a Rectangle in a Region
To determine if a specified point resides in a specified region, use XPointInRegion.

Bool XPointInRegion(r, x, y)
    Region r;
    int x, y;

r x y Specify the region, the x and y coordinates, which define the point.
The XPointInRegion function returns True if the point (x, y) is contained in the region r.

To determine if a specified rectangle is inside a region, use XRectInRegion.

int XRectInRegion(r, x, y, width, height)
    Region r;
    int x, y;
    unsigned int width, height;

r x y Specify the region, the x and y coordinates, which define the coordinates of the upper-left corner of the rectangle.

width height Specify the width and height, which define the rectangle.
The XRectInRegion function returns RectangleIn if the rectangle is entirely in the specified region, RectangleOut if the rectangle is entirely out of the specified region, and RectanglePart if the rectangle is partially in the specified region.

16.6. Using Cut Buffers
Xlib provides functions to manipulate cut buffers, a very simple form of "cut and paste" inter-client communication. Selections are a much more powerful and useful mechanism for interchanging data between clients (see section 4.5), and generally should be used instead of cut buffers.

Cut buffers are implemented as properties on the first root window of the display. The buffers can only contain text, in the STRING encoding. The text encoding is not changed by Xlib, when fetching or storing. Eight buffers are provided and can be accessed as a ring or as explicit buffers (numbered 0 through 7).

To store data in cut buffer 0, use XStoreBytes.
XStoreBytes(display, bytes, nbytes)
   Display *display;
   char *bytes;
   int nbytes;

display Specifies the connection to the X server.
bytes Specifies the bytes, which are not necessarily ASCII or null-terminated.
nbytes Specifies the number of bytes to be stored.

Note that the data can have embedded null characters, and need not be null terminated. The cut buffer’s contents can be retrieved later by any client calling XFetchBytes.

XStoreBytes can generate a BadAlloc error.

To store data in a specified cut buffer, use XStoreBuffer.

XStoreBuffer(display, bytes, nbytes, buffer)
   Display *display;
   char *bytes;
   int nbytes;
   int buffer;

display Specifies the connection to the X server.
bytes Specifies the bytes, which are not necessarily ASCII or null-terminated.
nbytes Specifies the number of bytes to be stored.
buffer Specifies the buffer in which you want to store the bytes.

If an invalid buffer is specified, the call has no effect. Note that the data can have embedded null characters, and need not be null terminated.

XStoreBuffer can generate a BadAlloc error.

To return data from cut buffer 0, use XFetchBytes.

char *XFetchBytes(display, nbytes_return)
   Display *display;
   int *nbytes_return;

display Specifies the connection to the X server.
nbytes_return Returns the number of bytes in the buffer.

The XFetchBytes function returns the number of bytes in the nbytes_return argument, if the buffer contains data. Otherwise, the function returns NULL and sets nbytes to 0. The appropriate amount of storage is allocated and the pointer returned. The client must free this storage when finished with it by calling XFree.

To return data from a specified cut buffer, use XFetchBuffer.

char *XFetchBuffer(display, nbytes_return, buffer)
   Display *display;
   int *nbytes_return;
   int buffer;

display Specifies the connection to the X server.
nbytes_return Returns the number of bytes in the buffer.
buffer Specifies the buffer from which you want the stored data returned.

The XFetchBuffer function returns zero to the nbytes_return argument if there is no data in the buffer or if an invalid buffer is specified.
To rotate the cut buffers, use \texttt{XRotateBuffers}.

\begin{verbatim}
XRotateBuffers(display, rotate)
    Display *display;
    int rotate;

    display Specifies the connection to the X server.
    rotate Specifies how much to rotate the cut buffers.
\end{verbatim}

The \texttt{XRotateBuffers} function rotates the cut buffers, such that buffer 0 becomes buffer \( n \), buffer 1 becomes \( n + 1 \mod 8 \), and so on. This cut buffer numbering is global to the display. Note that \texttt{XRotateBuffers} generates \texttt{BadMatch} errors if any of the eight buffers have not been created.

16.7. Determining the Appropriate Visual Type

A single display can support multiple screens. Each screen can have several different visual types supported at different depths. You can use the functions described in this section to determine which visual to use for your application.

The functions in this section use the visual information masks and the \texttt{XVisualInfo} structure, which is defined in \texttt{<X11/Xutil.h>} and contains:

\begin{verbatim}
/* Visual information mask bits */
#define VisualNoMask 0x0
#define VisualIDMask 0x1
#define VisualScreenWidthMask 0x2
#define VisualDepthMask 0x4
#define VisualClassMask 0x8
#define VisualRedMaskMask 0x10
#define VisualGreenMaskMask 0x20
#define VisualBlueMaskMask 0x40
#define VisualColorMapSizeMask 0x80
#define VisualBitsPerRGBMask 0x100
#define VisualAllMask 0xFF

/* Values */
typedef struct {
    Visual *visual;
    VisualID visualid;
    int screen;
    unsigned int depth;
    int class;
    unsigned long red_mask;
    unsigned long green_mask;
    unsigned long blue_mask;
    int colormap_size;
    int bits_per_rgb;
} XVisualInfo;
\end{verbatim}

To obtain a list of visual information structures that match a specified template, use \texttt{XGetVisualInfo}.
XVisualInfo *XGetVisualInfo(display, vinfo_mask, vinfo_template, nitems_return)
  Display *display;
  long vinfo_mask;
  XVisualInfo *vinfo_template;
  int *nitems_return;

display Specifies the connection to the X server.
vinfo_mask Specifies the visual mask value.
vinfo_template Specifies the visual attributes that are to be used in matching the visual structures.
nitems_return Returns the number of matching visual structures.

The XGetVisualInfo function returns a list of visual structures that have attributes equal to the attributes specified by vinfo_template. If no visual structures match the template using the specified vinfo_mask, XGetVisualInfo returns a NULL. To free the data returned by this function, use XFree.

To obtain the visual information that matches the specified depth and class of the screen, use XMatchVisualInfo.
Status XMatchVisualInfo(display, screen, depth, class, vinfo_return)
  Display *display;
  int screen;
  int depth;
  int class;
  XVisualInfo *vinfo_return;

display Specifies the connection to the X server.
screen Specifies the screen.
depth Specifies the depth of the screen.
class Specifies the class of the screen.
vinfo_return Returns the matched visual information.

The XMatchVisualInfo function returns the visual information for a visual that matches the specified depth and class for a screen. Because multiple visuals that match the specified depth and class can exist, the exact visual chosen is undefined. If a visual is found, XMatchVisualInfo returns nonzero and the information on the visual to vinfo_return. Otherwise, when a visual is not found, XMatchVisualInfo returns zero.

16.8. Manipulating Images
Xlib provides several functions that perform basic operations on images. All operations on images are defined using an XImage structure, as defined in <X11/Xlib.h>. Because the number of different types of image formats can be very large, this hides details of image storage properly from applications.

This section describes the functions for generic operations on images. Manufacturers can provide very fast implementations of these for the formats frequently encountered on their hardware. These functions are neither sufficient nor desirable to use for general image processing. Rather, they are here to provide minimal functions on screen format images. The basic operations for getting and putting images are XGetImage and XPutImage.

Note that no functions have been defined, as yet, to read and write images to and from disk files.

The XImage structure describes an image as it exists in the client's memory. The user can request that some of the members such as height, width, and xoffset be changed when the image is sent to the server. Note that bytes_per_line in concert with offset can be used to
extract a subset of the image. Other members (for example, byte order, bitmap_unit, and so forth) are characteristics of both the image and the server. If these members differ between the image and the server, XPutImage makes the appropriate conversions. The first byte of the first line of plane n must be located at the address (data + (n * height * bytes_per_line)). For a description of the XImage structure, see section 8.7.

To allocate sufficient memory for an XImage structure, use XCreateImage.

XImage *XCreateImage(display, visual, depth, format, offset, data, width, height, bitmap_pad, bytes_per_line)

Display *display;
Visual *visual;
unsigned int depth;
int format;
int offset;
char *data;
unsigned int width;
unsigned int height;
int bitmap_pad;
int bytes_per_line;

display Specifies the connection to the X server.
visual Specifies the Visual structure.
depth Specifies the depth of the image.
format Specifies the format for the image. You can pass XYBitmap, XYPixmap, or ZPixmap.
offset Specifies the number of pixels to ignore at the beginning of the scanline.
data Specifies the image data.
width Specifies the width of the image, in pixels.
height Specifies the height of the image, in pixels.
bitmap_pad Specifies the quantum of a scanline (8, 16, or 32). In other words, the start of one scanline is separated in client memory from the start of the next scanline by an integer multiple of this many bits.
bytes_per_line Specifies the number of bytes in the client image between the start of one scanline and the start of the next.

The XCreateImage function allocates the memory needed for an XImage structure for the specified display but does not allocate space for the image itself. Rather, it initializes the structure byte-order, bit-order, and bitmap-unit values from the display and returns a pointer to the XImage structure. The red, green, and blue mask values are defined for Z format images only and are derived from the Visual structure passed in. Other values also are passed in. The offset permits the rapid displaying of the image without requiring each scanline to be shifted into position. If you pass a zero value in bytes_per_line, Xlib assumes that the scanlines are contiguous in memory and calculates the value of bytes_per_line itself.

Note that when the image is created using XCreateImage, XGetImage, or XSubImage, the destroy procedure that the XDestroyImage function calls frees both the image structure and the data pointed to by the image structure.

The basic functions used to get a pixel, set a pixel, create a subimage, and add a constant value to an image are defined in the image object. The functions in this section are really macro invocations of the functions in the image object and are defined in <X11/Xutil.h>.

To obtain a pixel value in an image, use XGetPixel.
unsigned long XGetPixel(ximage, x, y)
    XImage *ximage;
    int x;
    int y;

    ximage Specifies the image.
    x
    y Specify the x and y coordinates.

The XGetPixel function returns the specified pixel from the named image. The pixel value is
returned in normalized format (that is, the least-significant byte of the long is the least-
significant byte of the pixel). The image must contain the x and y coordinates.

To set a pixel value in an image, use XPutPixel.

XPutPixel(ximage, x, y, pixel)
    XImage *ximage;
    int x;
    int y;
    unsigned long pixel;

    ximage Specifies the image.
    x
    y Specify the x and y coordinates.
    pixel Specifies the new pixel value.

The XPutPixel function overwrites the pixel in the named image with the specified pixel
value. The input pixel value must be in normalized format (that is, the least-significant byte of
the long is the least-significant byte of the pixel). The image must contain the x and y coordinates.

To create a subimage, use XSubImage.

XImage *XSubImage(ximage, x, y, subimage_width, subimage_height)
    XImage *ximage;
    int x;
    int y;
    unsigned int subimage_width;
    unsigned int subimage_height;

    ximage Specifies the image.
    x
    y Specify the x and y coordinates.

subimage_width Specify the width of the new subimage, in pixels.
subimage_height Specify the height of the new subimage, in pixels.

The XSubImage function creates a new image that is a subsection of an existing one. It allo-
cates the memory necessary for the new XImage structure and returns a pointer to the new
image. The data is copied from the source image, and the image must contain the rectangle
defined by x, y, subimage_width, and subimage_height.

To increment each pixel in an image by a constant value, use XAddPixel.

XAddPixel(ximage, value)
    XImage *ximage;
    long value;
The XSetImage function specifies an image. It is useful when you have a base pixel value from allocating color resources and need to manipulate the image to that form.

To deallocate the memory allocated in a previous call to XCreateImage, use XDestroyImage.

XDestroyImage(ximage)
   XImage *ximage;

ximage Specifies the image.

The XDestroyImage function deallocates the memory associated with the XImage structure. Note that when the image is created using XCreateImage, XGetImage, or XSubImage, the destroy procedure that this macro calls frees both the image structure and the data pointed to by the image structure.

16.9. Manipulating Bitmaps

Xlib provides functions that you can use to read a bitmap from a file, save a bitmap to a file, or create a bitmap. This section describes those functions that transfer bitmaps to and from the client's file system, thus allowing their reuse in a later connection (for example, from an entirely different client or to a different display or server).

The X version 11 bitmap file format is:

```
#define name_width width
#define name_height height
#define name_x_hot x
#define name_y_hot y
static unsigned char name_bits[] = { 0xAW,... }
```

The lines for the variables ending with _x_hot and _y_hot suffixes are optional because they are present only if a hotspot has been defined for this bitmap. The lines for the other variables are required. The word "unsigned" is optional; that is, the type of the _bits array can be char or unsigned char. The _bits array must be large enough to contain the size bitmap. The bitmap unit is eight. The name is derived from the name of the file that you specified on the original command line by deleting the directory path and extension.

To read a bitmap from a file, use XReadBitmapFile.

```
int XReadBitmapFile(display, d, filename, width_return, height_return, bitmap_return, x_hot_return, y_hot_return)
   Display *display;
   Drawable d;
   char *filename;
   unsigned int *width_return, *height_return;
   Pixmap *bitmap_return;
   int *x_hot_return, *y_hot_return;

   display Specifies the connection to the X server.
   d Specifies the drawable that indicates the screen.
   filename Specifies the file name to use. The format of the file name is operating-system dependent.
```

318
width_return
height_return
bitmap_return
x_hot_return
y_hot_return

Return the width and height values of the read in bitmap file.
Returns the bitmap that is created.
Return the hotspot coordinates.

The XReadBitmapFile function reads in a file containing a bitmap. The file is parsed in the encoding of the current locale. The ability to read other than the standard format is implementation dependent. If the file cannot be opened, XReadBitmapFile returns BitmapOpenFailed. If the file can be opened but does not contain valid bitmap data, it returns BitmapFileInvalid. If insufficient working storage is allocated, it returns BitmapNoMemory. If the file is readable and valid, it returns BitmapSuccess.

XReadBitmapFile returns the bitmap’s height and width, as read from the file, to width_return and height_return. It then creates a pixmap of the appropriate size, reads the bitmap data from the file into the pixmap, and assigns the pixmap to the caller’s variable bitmap. The caller must free the bitmap using XFreePixmap when finished. If name_x_hot and name_y_hot exist, XReadBitmapFile returns them to x_hot_return and y_hot_return; otherwise, it returns -1,-1.

XReadBitmapFile can generate BadAlloc and BadDrawable errors.

To write out a bitmap to a file, use XWriteBitmapFile.

int XWriteBitmapFile(display, filename, bitmap, width, height, x_hot, y_hot)

Display *display;
char *filename;
Pixmap bitmap;
unsigned int width, height;
int x_hot, y_hot;

display Specifies the connection to the X server.
filename Specifies the file name to use. The format of the file name is operating-system dependent.
bitmap Specifies the bitmap.
width Specify the width and height.
height
x_hot Specify where to place the hotspot coordinates (or -1,-1 if none are present) in the file.
y_hot

The XWriteBitmapFile function writes a bitmap out to a file in the X version 11 format. The file is written in the encoding of the current locale. If the file cannot be opened for writing, it returns BitmapOpenFailed. If insufficient memory is allocated, XWriteBitmapFile returns BitmapNoMemory; otherwise, on no error, it returns BitmapSuccess. If x_hot and y_hot are not -1, -1, XWriteBitmapFile writes them out as the hotspot coordinates for the bitmap.

XWriteBitmapFile can generate BadDrawable and BadMatch errors.

To create a pixmap and then store bitmap-format data into it, use XCreatePixmapFromBitmapData.
Pixmap XCreatePixmapFromBitmapData(display, d, data, width, height, fg, bg, depth)
  Display *display;
  Drawable *d;
  char *data;
  unsigned int width, height;
  unsigned long fg, bg;
  unsigned int depth;

  display Specifies the connection to the X server.
  d Specifies the drawable that indicates the screen.
  data Specifies the data in bitmap format.
  width Specify the width and height.
  height Specify the foreground and background pixel values to use.
  depth Specify the depth of the pixmap.

The XCreatePixmapFromBitmapData function creates a pixmap of the given depth and then
does a bitmap-format XPutImage of the data into it. The depth must be supported by the
screen of the specified drawable, or a BadMatch error results.

XCreatePixmapFromBitmapData can generate BadAlloc and BadMatch errors.

To include a bitmap written out by XWriteBitmapFile in a program directly, as opposed to
reading it in every time at run time, use XCreateBitmapFromData.

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

  display Specifies the connection to the X server.
  d Specifies the drawable that indicates the screen.
  data Specifies the location of the bitmap data.
  width Specify the width and height.
  height Specify the width and height.

The XCreateBitmapFromData function allows you to include in your C program (using
#include) a bitmap file that was written out by XWriteBitmapFile (X version 11 format only)
without reading in the bitmap file. The following example creates a gray bitmap:

#include "gray.bitmap"

Pixmap bitmap;
bitmap = XCreateBitmapFromData(display, window, gray_bits, gray_width, gray_height);

If insufficient working storage was allocated, XCreateBitmapFromData returns None. It is
your responsibility to free the bitmap using XFreePixmap when finished.

XCreateBitmapFromData can generate a BadAlloc error.

16.10. Using the Context Manager

The context manager provides a way of associating data with an X resource ID (mostly typi¬
cally a window) in your program. Note that this is local to your program; the data is not stored
in the server on a property list. Any amount of data in any number of pieces can be associated
with a resource ID, and each piece of data has a type associated with it. The context manager
requires knowledge of the resource ID and type to store or retrieve data.

Essentially, the context manager can be viewed as a two-dimensional, sparse array: one
dimension is subscripted by the X resource ID and the other by a context type field. Each
entry in the array contains a pointer to the data. Xlib provides context management functions
with which you can save data values, get data values, delete entries, and create a unique con¬
text type. The symbols used are in <X11/Xutil.h>.

To save a data value that corresponds to a resource ID and context type, use XSaveContext.

\[ \text{int XSaveContext(display, rid, context, data)} \]

- \text{display} Specifies the connection to the X server.
- \text{rid} Specifies the resource ID with which the data is associated.
- \text{context} Specifies the context type to which the data belongs.
- \text{data} Specifies the data to be associated with the window and type.

If an entry with the specified resource ID and type already exists, XSaveContext overrides it
with the specified context. The XSaveContext function returns a nonzero error code if an
error has occurred and zero otherwise. Possible errors are XCNOMEM (out of memory).

To get the data associated with a resource ID and type, use XFindContext.

\[ \text{int XFindContext(display, rid, context, data_return)} \]

- \text{display} Specifies the connection to the X server.
- \text{rid} Specifies the resource ID with which the data is associated.
- \text{context} Specifies the context type to which the data belongs.
- \text{data_return} Returns the data.

Because it is a return value, the data is a pointer. The XFindContext function returns a
nonzero error code if an error has occurred and zero otherwise. Possible errors are
XCNOENT (context-not-found).

To delete an entry for a given resource ID and type, use XDeleteContext.

\[ \text{int XDeleteContext(display, rid, context)} \]

- \text{display} Specifies the connection to the X server.
- \text{rid} Specifies the resource ID with which the data is associated.
- \text{context} Specifies the context type to which the data belongs.

The XDeleteContext function deletes the entry for the given resource ID and type from the
data structure. This function returns the same error codes that XFindContext returns if called
with the same arguments. XDeleteContext does not free the data whose address was saved.
To create a unique context type that may be used in subsequent calls to XSaveContext and XFindContext, use XUniqueContext.

XContext XUniqueContext()
Appendix A

Xlib Functions and Protocol Requests

This appendix provides two tables that relate to Xlib functions and the X protocol. The following table lists each Xlib function (in alphabetical order) and the corresponding protocol request that it generates.

<table>
<thead>
<tr>
<th>Xlib Function</th>
<th>Protocol Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>XActivateScreenSaver</td>
<td>ForceScreenSaver</td>
</tr>
<tr>
<td>XAddHost</td>
<td>ChangeHosts</td>
</tr>
<tr>
<td>XAddHosts</td>
<td>ChangeHosts</td>
</tr>
<tr>
<td>XAddToSaveSet</td>
<td>ChangeSaveSet</td>
</tr>
<tr>
<td>XAllocColor</td>
<td>AllocColor</td>
</tr>
<tr>
<td>XAllocColorCells</td>
<td>AllocColorCells</td>
</tr>
<tr>
<td>XAllocColorPlanes</td>
<td>AllocColorPlanes</td>
</tr>
<tr>
<td>XAllocNamedColor</td>
<td>AllocNamedColor</td>
</tr>
<tr>
<td>XAllowEvents</td>
<td>AllowEvents</td>
</tr>
<tr>
<td>XAutoRepeatOff</td>
<td>ChangeKeyboardControl</td>
</tr>
<tr>
<td>XAutoRepeatOn</td>
<td>ChangeKeyboardControl</td>
</tr>
<tr>
<td>XBell</td>
<td>Bell</td>
</tr>
<tr>
<td>XChangeActivePointerGrab</td>
<td>ChangeActivePointerGrab</td>
</tr>
<tr>
<td>XChangeGC</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XChangeKeyboardControl</td>
<td>ChangeKeyboardControl</td>
</tr>
<tr>
<td>XChangeKeyboardMapping</td>
<td>ChangeKeyboardMapping</td>
</tr>
<tr>
<td>XChangePointerControl</td>
<td>ChangePointerControl</td>
</tr>
<tr>
<td>XChangeProperty</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XChangeSaveSet</td>
<td>ChangeSaveSet</td>
</tr>
<tr>
<td>XChangeWindowAttributes</td>
<td>ChangeWindowAttributes</td>
</tr>
<tr>
<td>XCirculateSubwindows</td>
<td>CirculateWindow</td>
</tr>
<tr>
<td>XCirculateSubwindowsDown</td>
<td>CirculateWindow</td>
</tr>
<tr>
<td>XCirculateSubwindowsUp</td>
<td>CirculateWindow</td>
</tr>
<tr>
<td>XClearArea</td>
<td>ClearArea</td>
</tr>
<tr>
<td>XClearWindow</td>
<td>ClearArea</td>
</tr>
<tr>
<td>XConfigureWindow</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>XConvertSelection</td>
<td>ConvertSelection</td>
</tr>
<tr>
<td>XCopyArea</td>
<td>CopyArea</td>
</tr>
<tr>
<td>XCopyColormapAndFree</td>
<td>CopyColormapAndFree</td>
</tr>
<tr>
<td>XCopyGC</td>
<td>CopyGC</td>
</tr>
<tr>
<td>XCopyPlane</td>
<td>CopyPlane</td>
</tr>
<tr>
<td>XCreateBitmapFromData</td>
<td>CreatePixmap</td>
</tr>
<tr>
<td></td>
<td>FreeGC</td>
</tr>
<tr>
<td></td>
<td>PutImage</td>
</tr>
<tr>
<td>XCreateColormap</td>
<td>CreateColormap</td>
</tr>
<tr>
<td>XCreateFontCursor</td>
<td>CreateGlyphCursor</td>
</tr>
<tr>
<td>XCreateGC</td>
<td>CreateGC</td>
</tr>
<tr>
<td>XCreateGlyphCursor</td>
<td>CreateGlyphCursor</td>
</tr>
<tr>
<td>XCreatePixmap</td>
<td>CreatePixmap</td>
</tr>
<tr>
<td>XCreatePixmapCursor</td>
<td>CreateCursor</td>
</tr>
</tbody>
</table>

323
<table>
<thead>
<tr>
<th>Xlib Function</th>
<th>Protocol Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCreatePixmapFromData</td>
<td>CreateGC</td>
</tr>
<tr>
<td></td>
<td>CreatePixmap</td>
</tr>
<tr>
<td></td>
<td>FreeGC</td>
</tr>
<tr>
<td></td>
<td>PutImage</td>
</tr>
<tr>
<td>XCreateSimpleWindow</td>
<td>CreateWindow</td>
</tr>
<tr>
<td>XCreateWindow</td>
<td>CreateWindow</td>
</tr>
<tr>
<td>XDefineCursor</td>
<td>ChangeWindowAttributes</td>
</tr>
<tr>
<td>XDeleteProperty</td>
<td>DeleteProperty</td>
</tr>
<tr>
<td>XDestroySubwindows</td>
<td>DestroySubwindows</td>
</tr>
<tr>
<td>XDestroyWindow</td>
<td>DestroyWindow</td>
</tr>
<tr>
<td>XDisableAccessControl</td>
<td>SetAccessControl</td>
</tr>
<tr>
<td>XDrawArc</td>
<td>PolyArc</td>
</tr>
<tr>
<td>XDrawArcs</td>
<td>PolyArc</td>
</tr>
<tr>
<td>XDrawImageString</td>
<td>ImageText8</td>
</tr>
<tr>
<td>XDrawImageString16</td>
<td>ImageText16</td>
</tr>
<tr>
<td>XDrawLine</td>
<td>PolySegment</td>
</tr>
<tr>
<td>XDrawLines</td>
<td>PolyLine</td>
</tr>
<tr>
<td>XDrawPoint</td>
<td>PolyPoint</td>
</tr>
<tr>
<td>XDrawPoints</td>
<td>PolyPoint</td>
</tr>
<tr>
<td>XDrawRectangle</td>
<td>PolyRectangle</td>
</tr>
<tr>
<td>XDrawRectangles</td>
<td>PolyRectangle</td>
</tr>
<tr>
<td>XDrawSegments</td>
<td>PolySegment</td>
</tr>
<tr>
<td>XDrawString</td>
<td>PolyText8</td>
</tr>
<tr>
<td>XDrawString16</td>
<td>PolyText16</td>
</tr>
<tr>
<td>XDrawText</td>
<td>PolyText8</td>
</tr>
<tr>
<td>XDrawText16</td>
<td>PolyText16</td>
</tr>
<tr>
<td>XEnableAccessControl</td>
<td>SetAccessControl</td>
</tr>
<tr>
<td>XFetchBytes</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XFetchName</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XFillArc</td>
<td>PolyFillArc</td>
</tr>
<tr>
<td>XFillArcs</td>
<td>PolyFillArc</td>
</tr>
<tr>
<td>XFillPolygon</td>
<td>FillPoly</td>
</tr>
<tr>
<td>XFillRectangle</td>
<td>PolyFillRectangle</td>
</tr>
<tr>
<td>XFillRectangles</td>
<td>PolyFillRectangle</td>
</tr>
<tr>
<td>XForceScreenSaver</td>
<td>ForceScreenSaver</td>
</tr>
<tr>
<td>XFreeColormap</td>
<td>FreeColormap</td>
</tr>
<tr>
<td>XFreeColors</td>
<td>FreeColors</td>
</tr>
<tr>
<td>XFreeCursor</td>
<td>FreeCursor</td>
</tr>
<tr>
<td>XFreeFont</td>
<td>CloseFont</td>
</tr>
<tr>
<td>XFreeGC</td>
<td>FreeGC</td>
</tr>
<tr>
<td>XFreePixmap</td>
<td>FreePixmap</td>
</tr>
<tr>
<td>XGetAtomName</td>
<td>GetAtomName</td>
</tr>
<tr>
<td>XGetClassHint</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetFontPath</td>
<td>GetFontPath</td>
</tr>
<tr>
<td>XGetFontPath</td>
<td>GetFontPath</td>
</tr>
<tr>
<td>XGetGeometry</td>
<td>GetGeometry</td>
</tr>
<tr>
<td>XGetIconName</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetIconSizes</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetImage</td>
<td>GetImage</td>
</tr>
<tr>
<td>XGetInputFocus</td>
<td>GetInputFocus</td>
</tr>
<tr>
<td>XGetKeyboardControl</td>
<td>GetKeyboardControl</td>
</tr>
<tr>
<td>XGetKeyboardMapping</td>
<td>GetKeyboardMapping</td>
</tr>
<tr>
<td>XGetModifierMapping</td>
<td>GetModifierMapping</td>
</tr>
<tr>
<td>Xlib Function</td>
<td>Protocol Request</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
</tr>
<tr>
<td>XGetMotionEvents</td>
<td>GetMotionEvents</td>
</tr>
<tr>
<td>XGetModifierMapping</td>
<td>GetModifierMapping</td>
</tr>
<tr>
<td>XGetNormalHints</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetPointerControl</td>
<td>GetPointerControl</td>
</tr>
<tr>
<td>XGetPointerMapping</td>
<td>GetPointerMapping</td>
</tr>
<tr>
<td>XGetRGBColormaps</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetScreenSaver</td>
<td>GetScreenSaver</td>
</tr>
<tr>
<td>XGetSelectionOwner</td>
<td>GetSelectionOwner</td>
</tr>
<tr>
<td>XGetSizeHints</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetTextProperty</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetTransientForHint</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetWMClientMachine</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetWMColormapWindows</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetWMHints</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetWMIconName</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetWMName</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetWMNormalHints</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetWMProtocols</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetWMSizeHints</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetWMSizeHints</td>
<td>GetAtom</td>
</tr>
<tr>
<td>XGetWindowAttributes</td>
<td>GetWindowAttributes</td>
</tr>
<tr>
<td>XGetWindowProperty</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGetZoomHints</td>
<td>GetProperty</td>
</tr>
<tr>
<td>XGrabButton</td>
<td>GrabButton</td>
</tr>
<tr>
<td>XGrabKey</td>
<td>GrabKey</td>
</tr>
<tr>
<td>XGrabKeyboard</td>
<td>GrabKeyboard</td>
</tr>
<tr>
<td>XGrabPointer</td>
<td>GrabPointer</td>
</tr>
<tr>
<td>XGrabServer</td>
<td>GrabServer</td>
</tr>
<tr>
<td>XIconifyWindow</td>
<td>GetAtom</td>
</tr>
<tr>
<td>XInitExtension</td>
<td>QueryExtension</td>
</tr>
<tr>
<td>XInstallColormap</td>
<td>InstallColormap</td>
</tr>
<tr>
<td>XInternAtom</td>
<td>GetAtom</td>
</tr>
<tr>
<td>XKillClient</td>
<td>KillClient</td>
</tr>
<tr>
<td>XListExtensions</td>
<td>ListExtensions</td>
</tr>
<tr>
<td>XListFonts</td>
<td>ListFonts</td>
</tr>
<tr>
<td>XListFontsWithPathInfo</td>
<td>ListFontsWithPathInfo</td>
</tr>
<tr>
<td>XListHosts</td>
<td>ListHosts</td>
</tr>
<tr>
<td>XListInstalledColormaps</td>
<td>ListInstalledColormaps</td>
</tr>
<tr>
<td>XListProperties</td>
<td>ListProperties</td>
</tr>
<tr>
<td>XLoadFont</td>
<td>OpenFont</td>
</tr>
<tr>
<td>XLoadQueryFont</td>
<td>QueryFont</td>
</tr>
<tr>
<td>XLookupColor</td>
<td>LookupColor</td>
</tr>
<tr>
<td>XLowerWindow</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>XMapRaised</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>XMapWindow</td>
<td>MapWindow</td>
</tr>
<tr>
<td>XMapSubwindows</td>
<td>MapSubwindows</td>
</tr>
<tr>
<td>XSelectWindow</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>Xlib Function</td>
<td>Protocol Request</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>XMoveWindow</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>XNoOp</td>
<td>NoOperation</td>
</tr>
<tr>
<td>XOpenDisplay</td>
<td>CreateGC</td>
</tr>
<tr>
<td>XPARseColor</td>
<td>LookupColor</td>
</tr>
<tr>
<td>XPutImage</td>
<td>PutImage</td>
</tr>
<tr>
<td>XQueryBestCursor</td>
<td>QueryBestSize</td>
</tr>
<tr>
<td>XQueryBestSize</td>
<td>QueryBestSize</td>
</tr>
<tr>
<td>XQueryBestStipple</td>
<td>QueryBestSize</td>
</tr>
<tr>
<td>XQueryBestTile</td>
<td>QueryBestSize</td>
</tr>
<tr>
<td>XQueryColor</td>
<td>QueryColors</td>
</tr>
<tr>
<td>XQueryColors</td>
<td>QueryColors</td>
</tr>
<tr>
<td>XQueryExtension</td>
<td>QueryExtension</td>
</tr>
<tr>
<td>XQueryFont</td>
<td>QueryFont</td>
</tr>
<tr>
<td>XQueryKeymap</td>
<td>QueryKeymap</td>
</tr>
<tr>
<td>XQueryPointer</td>
<td>QueryPointer</td>
</tr>
<tr>
<td>XQueryTextExtents</td>
<td>QueryTextExtents</td>
</tr>
<tr>
<td>XQueryTextExtents16</td>
<td>QueryTextExtents</td>
</tr>
<tr>
<td>XQueryTree</td>
<td>QueryTree</td>
</tr>
<tr>
<td>XRaiseWindow</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>XReadBitmapFile</td>
<td>CreateGC</td>
</tr>
<tr>
<td></td>
<td>CreatePixmap</td>
</tr>
<tr>
<td></td>
<td>FreeGC</td>
</tr>
<tr>
<td>RecolorCursor</td>
<td>RecolorCursor</td>
</tr>
<tr>
<td>XReconfigureWMWindow</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>RemoveFromSaveSet</td>
<td>ChangeSaveSet</td>
</tr>
<tr>
<td>XRemoveHost</td>
<td>ChangeHosts</td>
</tr>
<tr>
<td>XRemoveHosts</td>
<td>ChangeHosts</td>
</tr>
<tr>
<td>ReparentWindow</td>
<td>ReparentWindow</td>
</tr>
<tr>
<td>XResetScreenSaver</td>
<td>ForceScreenSaver</td>
</tr>
<tr>
<td>ResizeWindow</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>RestackWindows</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>XRotateBuffers</td>
<td>RotateProperties</td>
</tr>
<tr>
<td>XRotateWindowProperties</td>
<td>RotateProperties</td>
</tr>
<tr>
<td>XSelectInput</td>
<td>ChangeWindowAttributes</td>
</tr>
<tr>
<td>SendEvent</td>
<td>SendEvent</td>
</tr>
<tr>
<td>XSetAccessControl</td>
<td>SetAccessControl</td>
</tr>
<tr>
<td>XSetArcMode</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetBackgroundColor</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetClassHint</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetClipMask</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetClipOrigin</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetClipRectangles</td>
<td>SetClipRectangles</td>
</tr>
<tr>
<td>XSetCloseDownMode</td>
<td>SetCloseDownMode</td>
</tr>
<tr>
<td>XSetCommand</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetDashes</td>
<td>SetDashes</td>
</tr>
<tr>
<td>XSetFillRule</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetFillStyle</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetFont</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetFontPath</td>
<td>SetFontPath</td>
</tr>
<tr>
<td>XSetForeground</td>
<td>ChangeGC</td>
</tr>
</tbody>
</table>

326
<table>
<thead>
<tr>
<th>Xlib Function</th>
<th>Protocol Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSetFunction</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetGraphicsExposures</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetIconName</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetIconSizes</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetInputFocus</td>
<td>SetInputFocus</td>
</tr>
<tr>
<td>XSetLineAttributes</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetModifierMapping</td>
<td>SetModifierMapping</td>
</tr>
<tr>
<td>XSetNormalHints</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetPlaneMask</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetPointerMapping</td>
<td>SetPointerMapping</td>
</tr>
<tr>
<td>XSetRGBColormaps</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetScreenSaver</td>
<td>SetScreenSaver</td>
</tr>
<tr>
<td>XSetSelectionOwner</td>
<td>SetSelectionOwner</td>
</tr>
<tr>
<td>XSetSizeHints</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetStandardProperties</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetState</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetStipple</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetSubwindowMode</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetTextProperty</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetTile</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetTransientForHint</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetTOSOrigin</td>
<td>ChangeGC</td>
</tr>
<tr>
<td>XSetWMClientMachine</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetWMColormapWindows</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetWMHints</td>
<td>InternAtom</td>
</tr>
<tr>
<td>XSetWMIconName</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetWMName</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetWMNormalHints</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetWMProperties</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetWMProtocols</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetWSWindowHints</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XSetWindowBackground</td>
<td>ChangeWindowAttributes</td>
</tr>
<tr>
<td>XSetWindowBackgroundPixmap</td>
<td>ChangeWindowAttributes</td>
</tr>
<tr>
<td>XSetWindowBorder</td>
<td>ChangeWindowAttributes</td>
</tr>
<tr>
<td>XSetWindowBorderPixmap</td>
<td>ChangeWindowAttributes</td>
</tr>
<tr>
<td>XSetWindowBorderWidth</td>
<td>ConfigureWindow</td>
</tr>
<tr>
<td>XSetWindowColormap</td>
<td>ChangeWindowAttributes</td>
</tr>
<tr>
<td>XSetZoomHints</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XStoreBuffer</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XStoreBytes</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XStoreColor</td>
<td>StoreColors</td>
</tr>
<tr>
<td>XStoreColors</td>
<td>StoreColors</td>
</tr>
<tr>
<td>XStoreName</td>
<td>ChangeProperty</td>
</tr>
<tr>
<td>XStoreNamedColor</td>
<td>StoreNamedColor</td>
</tr>
<tr>
<td>XSync</td>
<td>GetInputFocus</td>
</tr>
<tr>
<td>XSynchronize</td>
<td>GetInputFocus</td>
</tr>
<tr>
<td>XTranslateCoordinates</td>
<td>TranslateCoordinates</td>
</tr>
<tr>
<td>XUndefineCursor</td>
<td>ChangeWindowAttributes</td>
</tr>
<tr>
<td>XUngrabButton</td>
<td>UngrabButton</td>
</tr>
<tr>
<td>XUngrabKey</td>
<td>UngrabKey</td>
</tr>
<tr>
<td>Xlib Function</td>
<td>Protocol Request</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>XUngrabKeyboard</td>
<td>UngrabKeyboard</td>
</tr>
<tr>
<td>XUngrabPointer</td>
<td>UngrabPointer</td>
</tr>
<tr>
<td>XUngrabServer</td>
<td>UngrabServer</td>
</tr>
<tr>
<td>XUninstallColormap</td>
<td>UninstallColormap</td>
</tr>
<tr>
<td>XUnloadFont</td>
<td>CloseFont</td>
</tr>
<tr>
<td>XUnmapSubwindows</td>
<td>UnmapSubwindows</td>
</tr>
<tr>
<td>XUnmapWindow</td>
<td>UnmapWindow</td>
</tr>
<tr>
<td>XWarpPointer</td>
<td>WarpPointer</td>
</tr>
<tr>
<td>XWithdrawWindow</td>
<td>SendEvent</td>
</tr>
<tr>
<td></td>
<td>UnmapWindow</td>
</tr>
</tbody>
</table>
The following table lists each X protocol request (in alphabetical order) and the Xlib functions that reference it.

<table>
<thead>
<tr>
<th>Protocol Request</th>
<th>Xlib Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllocColor</td>
<td>XAllocColor</td>
</tr>
<tr>
<td>AllocColorCells</td>
<td>XAllocColorCells</td>
</tr>
<tr>
<td>AllocColorPlanes</td>
<td>XAllocColorPlanes</td>
</tr>
<tr>
<td>AllocNamedColor</td>
<td>XAllocNamedColor</td>
</tr>
<tr>
<td>AllowEvents</td>
<td>XAllowEvents</td>
</tr>
<tr>
<td>Bell</td>
<td>XBell</td>
</tr>
<tr>
<td>SetAccessControl</td>
<td>XDisableAccessControl</td>
</tr>
<tr>
<td></td>
<td>XEnableAccessControl</td>
</tr>
<tr>
<td></td>
<td>XSetAccessControl</td>
</tr>
<tr>
<td>ChangeActivePointerGrab</td>
<td>XChangeActivePointerGrab</td>
</tr>
<tr>
<td>SetCloseDownMode</td>
<td>XSetCloseDownMode</td>
</tr>
<tr>
<td>ChangeGC</td>
<td>XChangeGC</td>
</tr>
<tr>
<td></td>
<td>XSetArcMode</td>
</tr>
<tr>
<td></td>
<td>XSetBackground</td>
</tr>
<tr>
<td></td>
<td>XSetClipMask</td>
</tr>
<tr>
<td></td>
<td>XSetClipOrigin</td>
</tr>
<tr>
<td></td>
<td>XSetFillRule</td>
</tr>
<tr>
<td></td>
<td>XSetFillStyle</td>
</tr>
<tr>
<td></td>
<td>XSetFont</td>
</tr>
<tr>
<td></td>
<td>XSetForeground</td>
</tr>
<tr>
<td></td>
<td>XSetFunction</td>
</tr>
<tr>
<td></td>
<td>XSetGraphicsExposures</td>
</tr>
<tr>
<td></td>
<td>XSetLineAttributes</td>
</tr>
<tr>
<td></td>
<td>XSetPlaneMask</td>
</tr>
<tr>
<td></td>
<td>XSetState</td>
</tr>
<tr>
<td></td>
<td>XSetStipple</td>
</tr>
<tr>
<td></td>
<td>XSetSubwindowMode</td>
</tr>
<tr>
<td></td>
<td>XSetTile</td>
</tr>
<tr>
<td></td>
<td>XSetTSOrigin</td>
</tr>
<tr>
<td>ChangeHosts</td>
<td>XAddHost</td>
</tr>
<tr>
<td></td>
<td>XAddHosts</td>
</tr>
<tr>
<td></td>
<td>XRemoveHost</td>
</tr>
<tr>
<td></td>
<td>XRemoveHosts</td>
</tr>
<tr>
<td>ChangeKeyboardControl</td>
<td>XAutoRepeatOff</td>
</tr>
<tr>
<td></td>
<td>XAutoRepeatOn</td>
</tr>
<tr>
<td></td>
<td>XChangeKeyboardControl</td>
</tr>
<tr>
<td>ChangeKeyboardMapping</td>
<td>XChangeKeyboardMapping</td>
</tr>
<tr>
<td>ChangePointerControl</td>
<td>XChangePointerControl</td>
</tr>
<tr>
<td>ChangeProperty</td>
<td>XChangeProperty</td>
</tr>
<tr>
<td></td>
<td>XSetClassHint</td>
</tr>
<tr>
<td></td>
<td>XSetCommand</td>
</tr>
<tr>
<td></td>
<td>XSetIconName</td>
</tr>
<tr>
<td></td>
<td>XSetIconSizes</td>
</tr>
<tr>
<td></td>
<td>XSetNormalHints</td>
</tr>
<tr>
<td></td>
<td>XSetRGBColormaps</td>
</tr>
<tr>
<td></td>
<td>XSetSizeHints</td>
</tr>
<tr>
<td></td>
<td>XSetStandardProperties</td>
</tr>
<tr>
<td></td>
<td>XSetTextProperty</td>
</tr>
<tr>
<td></td>
<td>XSetTransientsForHint</td>
</tr>
</tbody>
</table>

329
<table>
<thead>
<tr>
<th>Protocol Request</th>
<th>Xlib Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeSaveSet</td>
<td>XAddToSaveSet</td>
</tr>
<tr>
<td>ChangeWindowAttributes</td>
<td>XChangeWindowAttributes</td>
</tr>
<tr>
<td>CirculateWindow</td>
<td>XCirculateSubwindowsDown</td>
</tr>
<tr>
<td>ClearArea</td>
<td>XClearArea</td>
</tr>
<tr>
<td>CloseFont</td>
<td>XFreeFont</td>
</tr>
<tr>
<td>ConfigureWindow</td>
<td>XConfigureWindow</td>
</tr>
<tr>
<td>ConvertSelection</td>
<td>XConvertSelection</td>
</tr>
<tr>
<td>CopyArea</td>
<td>XCopyArea</td>
</tr>
<tr>
<td>CopyColormapAndFree</td>
<td>XCopyColormapAndFree</td>
</tr>
<tr>
<td>CopyGC</td>
<td>XCopyGC</td>
</tr>
<tr>
<td>CopyPlane</td>
<td>XCopyPlane</td>
</tr>
<tr>
<td>CreateColormap</td>
<td>XCreateColormap</td>
</tr>
<tr>
<td>CreateCursor</td>
<td>XCreatePixmapCursor</td>
</tr>
<tr>
<td>CreateGC</td>
<td>XCreateGC</td>
</tr>
<tr>
<td></td>
<td>XCreateBitmapFromData</td>
</tr>
<tr>
<td></td>
<td>XCreatePixmapFromData</td>
</tr>
<tr>
<td>Protocol Request</td>
<td>Xlib Function</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>CreateGlyphCursor</td>
<td>XCreatePixmap</td>
</tr>
<tr>
<td>CreatePixmap</td>
<td>XCreatePixmap</td>
</tr>
<tr>
<td>CreateWindow</td>
<td>XCreateSimpleWindow</td>
</tr>
<tr>
<td>DeleteProperty</td>
<td>XDeleteProperty</td>
</tr>
<tr>
<td>DestroySubwindows</td>
<td>XDestroySubwindows</td>
</tr>
<tr>
<td>DestroyWindow</td>
<td>XDestroyWindow</td>
</tr>
<tr>
<td>FillPoly</td>
<td>XFillPolygon</td>
</tr>
<tr>
<td>ForceScreenSaver</td>
<td>XActivateScreenSaver</td>
</tr>
<tr>
<td>FreeColormap</td>
<td>XFreeColormap</td>
</tr>
<tr>
<td>FreeColors</td>
<td>XFreeColors</td>
</tr>
<tr>
<td>FreeCursor</td>
<td>XFreeCursor</td>
</tr>
<tr>
<td>FreeGC</td>
<td>XFreeGC</td>
</tr>
<tr>
<td>FreePixmap</td>
<td>XFreePixmap</td>
</tr>
<tr>
<td>GetAtomName</td>
<td>XGetAtomName</td>
</tr>
<tr>
<td>GetFontPath</td>
<td>XGetFontPath</td>
</tr>
<tr>
<td>GetGeometry</td>
<td>XGetGeometry</td>
</tr>
<tr>
<td>GetImage</td>
<td>XGetImage</td>
</tr>
<tr>
<td>GetInputFocus</td>
<td>XGetInputFocus</td>
</tr>
<tr>
<td>GetPointerControl</td>
<td>XGetPointerControl</td>
</tr>
<tr>
<td>GetPointerMapping</td>
<td>XGetPointerMapping</td>
</tr>
<tr>
<td>GetProperty</td>
<td>XFetchBytes</td>
</tr>
<tr>
<td></td>
<td>XFetchName</td>
</tr>
<tr>
<td></td>
<td>XGetClassHint</td>
</tr>
<tr>
<td></td>
<td>XGetIconName</td>
</tr>
<tr>
<td></td>
<td>XGetIconSizes</td>
</tr>
<tr>
<td></td>
<td>XGetNormalHints</td>
</tr>
<tr>
<td></td>
<td>XGetRGBColormaps</td>
</tr>
<tr>
<td></td>
<td>XGetSizeHints</td>
</tr>
<tr>
<td></td>
<td>XGetTextProperty</td>
</tr>
<tr>
<td></td>
<td>XGetTransientForHint</td>
</tr>
<tr>
<td></td>
<td>XGetWMClientMachine</td>
</tr>
<tr>
<td></td>
<td>XGetWMColormapWindows</td>
</tr>
<tr>
<td></td>
<td>XGetWMHints</td>
</tr>
<tr>
<td>Protocol Request</td>
<td>Xlib Function</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>GetSelectionOwner</td>
<td>XGetSelectionOwner</td>
</tr>
<tr>
<td>GetWindowAttributes</td>
<td>XGetWindowAttributes</td>
</tr>
<tr>
<td>GrabButton</td>
<td>XGrabButton</td>
</tr>
<tr>
<td>GrabKey</td>
<td>XGrabKey</td>
</tr>
<tr>
<td>GrabKeyboard</td>
<td>XGrabKeyboard</td>
</tr>
<tr>
<td>GrabPointer</td>
<td>XGrabPointer</td>
</tr>
<tr>
<td>GrabServer</td>
<td>XGrabServer</td>
</tr>
<tr>
<td>ImageText16</td>
<td>XDrawImageString16</td>
</tr>
<tr>
<td>ImageText8</td>
<td>XDrawImageString</td>
</tr>
<tr>
<td>InstallColormap</td>
<td>XInstallColormap</td>
</tr>
<tr>
<td>InternAtom</td>
<td>XGetWMColormapWindows</td>
</tr>
<tr>
<td></td>
<td>XGetWMProtocols</td>
</tr>
<tr>
<td></td>
<td>XConfitWindow</td>
</tr>
<tr>
<td></td>
<td>XInternAtom</td>
</tr>
<tr>
<td></td>
<td>XSetWMColormapWindows</td>
</tr>
<tr>
<td></td>
<td>XSetWMProtocols</td>
</tr>
<tr>
<td>KillClient</td>
<td>XKillClient</td>
</tr>
<tr>
<td>ListExtensions</td>
<td>XListExtensions</td>
</tr>
<tr>
<td>ListFonts</td>
<td>XListFonts</td>
</tr>
<tr>
<td>ListFontsWithInfo</td>
<td>XListFontsWithInfo</td>
</tr>
<tr>
<td>ListHosts</td>
<td>XListHosts</td>
</tr>
<tr>
<td>ListInstalledColormaps</td>
<td>XListInstalledColormaps</td>
</tr>
<tr>
<td>ListProperties</td>
<td>XListProperties</td>
</tr>
<tr>
<td>LookupColor</td>
<td>XLookupColor</td>
</tr>
<tr>
<td></td>
<td>XPARseColor</td>
</tr>
<tr>
<td>MapSubwindows</td>
<td>XMapSubwindows</td>
</tr>
<tr>
<td>MapWindow</td>
<td>XMapRaised</td>
</tr>
<tr>
<td></td>
<td>XMapWindow</td>
</tr>
<tr>
<td>NoOperation</td>
<td>XNoOp</td>
</tr>
<tr>
<td>OpenFont</td>
<td>XLoadFont</td>
</tr>
<tr>
<td></td>
<td>XLoadQueryFont</td>
</tr>
<tr>
<td>PolyArc</td>
<td>XDrawArc</td>
</tr>
<tr>
<td></td>
<td>XDrawArcs</td>
</tr>
<tr>
<td>PolyFillArc</td>
<td>XFillArc</td>
</tr>
<tr>
<td></td>
<td>XFillArcs</td>
</tr>
<tr>
<td>PolyFillRectangle</td>
<td>XFillRectangle</td>
</tr>
<tr>
<td></td>
<td>XFllRectangles</td>
</tr>
<tr>
<td>PolyLine</td>
<td>XDrawLines</td>
</tr>
<tr>
<td>PolyPoint</td>
<td>XDrawPoint</td>
</tr>
<tr>
<td></td>
<td>XDrawPoints</td>
</tr>
<tr>
<td>PolyRectangle</td>
<td>XDrawRectangle</td>
</tr>
<tr>
<td></td>
<td>XDrawRectangles</td>
</tr>
<tr>
<td>PolySegment</td>
<td>XDrawLine</td>
</tr>
<tr>
<td></td>
<td>XDrawSegments</td>
</tr>
<tr>
<td>PolyText16</td>
<td>XDrawString16</td>
</tr>
<tr>
<td>Protocol Request</td>
<td>Xlib Function</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>PolyText8</td>
<td>XDrawText16</td>
</tr>
<tr>
<td></td>
<td>XDrawString</td>
</tr>
<tr>
<td></td>
<td>XDrawText</td>
</tr>
<tr>
<td></td>
<td>XPutImage</td>
</tr>
<tr>
<td></td>
<td>XCreateBitmapFromData</td>
</tr>
<tr>
<td></td>
<td>XCreatePixmapFromData</td>
</tr>
<tr>
<td></td>
<td>XReadBitmapFile</td>
</tr>
<tr>
<td>QueryBestSize</td>
<td>XQueryBestCursor</td>
</tr>
<tr>
<td></td>
<td>XQueryBestSize</td>
</tr>
<tr>
<td></td>
<td>XQueryBestStipple</td>
</tr>
<tr>
<td></td>
<td>XQueryBestTile</td>
</tr>
<tr>
<td>QueryColors</td>
<td>XQueryColor</td>
</tr>
<tr>
<td></td>
<td>XQueryColors</td>
</tr>
<tr>
<td>QueryExtension</td>
<td>XInitExtension</td>
</tr>
<tr>
<td></td>
<td>XQueryExtension</td>
</tr>
<tr>
<td>QueryFont</td>
<td>XLoadQueryFont</td>
</tr>
<tr>
<td></td>
<td>XQueryFont</td>
</tr>
<tr>
<td>QueryKeymap</td>
<td>XQueryKeymap</td>
</tr>
<tr>
<td>QueryPointer</td>
<td>XQueryPointer</td>
</tr>
<tr>
<td>QueryTextExtents</td>
<td>XQueryTextExtents</td>
</tr>
<tr>
<td></td>
<td>XQueryTextExtents16</td>
</tr>
<tr>
<td>QueryTree</td>
<td>XQueryTree</td>
</tr>
<tr>
<td>RecolorCursor</td>
<td>XRecolorCursor</td>
</tr>
<tr>
<td>ReparentWindow</td>
<td>XRepaintWindow</td>
</tr>
<tr>
<td>RotateProperties</td>
<td>XRotateBuffers</td>
</tr>
<tr>
<td></td>
<td>XRotateWindowProperties</td>
</tr>
<tr>
<td>SendEvent</td>
<td>XIConifyWindow</td>
</tr>
<tr>
<td></td>
<td>XReconfigureWMWindow</td>
</tr>
<tr>
<td></td>
<td>XSendEvent</td>
</tr>
<tr>
<td></td>
<td>XWithdrawWindow</td>
</tr>
<tr>
<td>SetClipRectangles</td>
<td>XSetClipRectangles</td>
</tr>
<tr>
<td>SetCloseDownMode</td>
<td>XSetCloseDownMode</td>
</tr>
<tr>
<td>SetDashes</td>
<td>XSetDashes</td>
</tr>
<tr>
<td>SetFontPath</td>
<td>XSetFontPath</td>
</tr>
<tr>
<td>SetInputFocus</td>
<td>XSetInputFocus</td>
</tr>
<tr>
<td>SetModifierMapping</td>
<td>XSetModifierMapping</td>
</tr>
<tr>
<td>SetPointerMapping</td>
<td>XSetPointerMapping</td>
</tr>
<tr>
<td>SetScreenSaver</td>
<td>XGetScreenSaver</td>
</tr>
<tr>
<td></td>
<td>XSetScreenSaver</td>
</tr>
<tr>
<td>SetSelectionOwner</td>
<td>XSetSelectionOwner</td>
</tr>
<tr>
<td>StoreColors</td>
<td>XStoreColor</td>
</tr>
<tr>
<td></td>
<td>XStoreColors</td>
</tr>
<tr>
<td>StoreNamedColor</td>
<td>XStoreNamedColor</td>
</tr>
<tr>
<td>TranslateCoordinates</td>
<td>XTranslateCoordinates</td>
</tr>
<tr>
<td>UngrabButton</td>
<td>XUngrabButton</td>
</tr>
<tr>
<td>UngrabKey</td>
<td>XUngrabKey</td>
</tr>
<tr>
<td>UngrabKeyboard</td>
<td>XUngrabKeyboard</td>
</tr>
<tr>
<td>UngrabPointer</td>
<td>XUngrabPointer</td>
</tr>
<tr>
<td>UngrabServer</td>
<td>XUngrabServer</td>
</tr>
<tr>
<td>UninstallColormap</td>
<td>XUninstallColormap</td>
</tr>
<tr>
<td>UnmapSubwindows</td>
<td>XUnmapSubWindows</td>
</tr>
<tr>
<td>UnmapWindow</td>
<td>XUnmapWindow</td>
</tr>
</tbody>
</table>

333
<table>
<thead>
<tr>
<th>Protocol Request</th>
<th>Xlib Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>WarpPointer</td>
<td>XWithdrawWindow</td>
</tr>
<tr>
<td></td>
<td>XWarpPointer</td>
</tr>
</tbody>
</table>
The following are the available cursors that can be used with XCreateFontCursor:

```c
#define XC_X_cursor 0
#define XC_arrow 2
#define XC_based_arrow_down 4
#define XC_based_arrow_up 6
#define XC_boat 8
#define XC_bogosity 10
#define XC_bottom_left_corner 12
#define XC_bottom_right_corner 14
#define XC_bottom_side 16
#define XC_bottom_tee 18
#define XC_center_ptr 22
#define XC_circle 24
#define XC_clock 26
#define XC_dodger_box 28
#define XC_dodges_box_mask 30
#define XC_double_arrow 32
#define XC_draft_large 34
#define XC_draft_small 36
#define XC_draped_box 38
#define XC_dot 40
#define XC_dot_box_mask 42
#define XC_dot_bracket_mask 44
#define XC_dot_bracket_mask_1 46
#define XC_dotted_cross 48
#define XC_exchange 50
#define XC_exclamation_point 52
#define XC_gaming 54
#define XC_gumby 56
#define XC_heart 58
#define XC_icon 60
#define XC_icon_64 62
#define XC_iron_cross 64
#define XC_left_ptr 66
#define XC_left_side 68
#define XC_left_tee 70
#define XC_leftbutton 72
#define XC_question_arrow 74
#define XCロー 76
#define XC_left_tee 78
#define XC_magnifying_glass 80
#define XC_magnifying_glass_82
#define XC_MOUSE 84
#define XC_pencil 86
#define XC_pirate 88
#define XC_plus 90
#define XC_question_arrow 92
#define XC_right_ptr 94
#define XC_right_side 96
#define XC_right_tee 98
#define XC_rightbutton 100
#define XC_rtl_logo 102
#define XC_sailboat 104
#define XC_left_down_arrow 106
#define XC_left_h_double_arrow 108
#define XC_left_arrow 110
#define XC_left_right_arrow 112
#define XC_left_up_arrow 114
#define XC_left_double_arrow 116
#define XC_shuttle 118
#define XC_sizing 120
#define XC_spider 122
#define XC_spraying_can 124
#define XC_star 126
#define XC_test 128
#define XC_tcross 130
#define XC_top_left_arrow 132
#define XC_top_left_corner 134
#define XC_top_right_corner 136
#define XC_top_side 138
#define XC_top_tee 140
#define XC_trek 142
#define XC_ul_angle 144
#define XC_umbrella 146
#define XC_url 148
#define XC_watch 150
#define XC_xterm 152
```
Appendix C
Extensions

Because X can evolve by extensions to the core protocol, it is important that extensions not be perceived as second class citizens. At some point, your favorite extensions may be adopted as additional parts of the X Standard.

Therefore, there should be little to distinguish the use of an extension from that of the core protocol. To avoid having to initialize extensions explicitly in application programs, it is also important that extensions perform "lazy evaluations" and automatically initialize themselves when called for the first time.

This appendix describes techniques for writing extensions to Xlib that will run at essentially the same performance as the core protocol requests.

Note

It is expected that a given extension to X consists of multiple requests. Defining ten new features as ten separate extensions is a bad practice. Rather, they should be packaged into a single extension and should use minor opcodes to distinguish the requests.

The symbols and macros used for writing stubs to Xlib are listed in <XI1/Xlibint.h>.

Basic Protocol Support Routines

The basic protocol requests for extensions are XQueryExtension and XListExtensions.

`Bool XQueryExtension_DISPLAY name major_opcode_return first_event_return first_error_return`

- `DISPLAY display` Specifies the connection to the X server.
- `name char *name;` Specifies the extension name.
- `major_opcode_return int *major_opcode_return;` Returns the major opcode.
- `first_event_return int *first_event_return;` Returns the first event code, if any.
- `first_error_return` Specifies the extension list.

`XQueryExtension` determines if the named extension is present. If the extension is not present, False is returned; otherwise True is returned. If the extension is present, the major opcode for the extension is returned to major_opcode_return; 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 to first_event_return; 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 to first_error_return; otherwise, zero is returned. The format of additional data in the errors is specific to the extension.

If the extension name is not in the Host Portable Character Encoding the result is implementation dependent. Case matters; the strings `thing`, `Thing`, and `thinG` are all considered different
names.

char **XListExtensions(display, nextensions_return)
    Display *display;
    int *nextensions_return;

display Specifies the connection to the X server.
nextensions_return Returns the number of extensions listed.

XListExtensions returns a list of all extensions supported by the server. If the data returned
by the server is in the Latin Portable Character Encoding, then the returned strings are in the
Host Portable Character Encoding. Otherwise, the result is implementation dependent.

XFreeExtensionList(list)
    char **list;

list Specifies the list of extension names.

XFreeExtensionList frees the memory allocated by XListExtensions.

Hooking into Xlib

These functions allow you to hook into the library. They are not normally used by application
programmers but are used by people who need to extend the core X protocol and the X library
interface. The functions, which generate protocol requests for X, are typically called stubs.

In extensions, stubs first should check to see if they have initialized themselves on a connec-
tion. If they have not, they then should call XInitExtension to attempt to initialize themselves
on the connection.

If the extension needs to be informed of GC/font allocation or deallocation or if the extension
defines new event types, the functions described here allow the extension to be called when
these events occur.

The XExtCodes structure returns the information from XInitExtension and is defined in
<X11/Xlib.h>:

typedef struct _XExtCodes {
    int extension;  /* public to extension, cannot be changed */
    int major_opcode;  /* extension number */
    int first_event;  /* major op-code assigned by server */
    int first_error;  /* first event number for the extension */
    int first_error;  /* first error number for the extension */
} XExtCodes;

XExtCodes *XInitExtension(display, name)
    Display *display;
    char *name;

display Specifies the connection to the X server.
name Specifies the extension name.

XInitExtension determines if the named extension exists. Then, it allocates storage for main-
taining the information about the extension on the connection, chains this onto the extension
list for the connection, and returns the information the stub implementor will need to access the
extension. If the extension does not exist, XInitExtension returns NULL.

If the extension name is not in the Host Portable Character Encoding the result is implementa-
tion dependent. Case matters; the strings thing, Thing, and thinG are all considered different
names.

The extension number in the XExtCodes structure is needed in the other calls that follow.
This extension number is unique only to a single connection.
XExtCodes *XAddExtension(display)
   Display *display;

display       Specifies the connection to the X server.

For local Xlib extensions, XAddExtension allocates the XExtCodes structure, bumps the
extension number count, and chains the extension onto the extension list. (This permits exten-
sions to Xlib without requiring server extensions.)

Hooks into the Library

These functions allow you to define procedures that are to be called when various cir-
sumstances occur. The procedures include the creation of a new GC for a connection, the
copying of a GC, the freeing of a GC, the creating and freeing of fonts, the conversion of
events defined by extensions to and from wire format, and the handling of errors.

All of these functions return the previous routine defined for this extension.

int (*XESetCloseDisplay(display, extension, proc))()
   Display *display;
   int extension;
   int (*proc)();

display       Specifies the connection to the X server.
extension     Specifies the extension number.
proc          Specifies the routine to call when the display is closed.

You use this procedure to define a procedure to be called whenever XCloseDisplay is called.
This procedure returns any previously defined procedure, usually NULL.

When XCloseDisplay is called, your routine is called with these arguments:

(*proc)(display, codes)
   Display *display;
   XExtCodes *codes;

int (*XESetCreateGC(display, extension, proc))()
   Display *display;
   int extension;
   int (*proc)();

display       Specifies the connection to the X server.
extension     Specifies the extension number.
proc          Specifies the routine to call when a GC is closed.

You use this procedure to define a procedure to be called whenever a new GC is created. This
procedure returns any previously defined procedure, usually NULL.

When a GC is created, your routine is called with these arguments:

(*proc)(display, gc, codes)
   Display *display;
   GC gc;
   XExtCodes *codes;

int (*XESetCopyGC(display, extension, proc))()
   Display *display;
   int extension;
   int (*proc)();

display       Specifies the connection to the X server.
extension Specifies the extension number.

proc Specifies the routine to call when GC components are copied.

You use this procedure to define a procedure to be called whenever a GC is copied. This procedure returns any previously defined procedure, usually NULL.

When a GC is copied, your routine is called with these arguments:

\[(\text{proc})(\text{display}, \text{gc}, \text{codes})\]

\[
\begin{align*}
\text{Display} & \quad \text{*display}; \\
\text{GC} & \quad \text{gc}; \\
\text{XExtCodes} & \quad *\text{codes};
\end{align*}
\]

```c
int (*XESetFreeGC(display, extension, proc))()
\
\begin{align*}
\text{Display} & \quad *\text{display}; \\
\text{int} & \quad \text{extension}; \\
\text{int} & \quad (*\text{proc})();
\end{align*}
```

\textit{display} Specifies the connection to the X server.

\textit{extension} Specifies the extension number.

\textit{proc} Specifies the routine to call when a GC is freed.

You use this procedure to define a procedure to be called whenever a GC is freed. This procedure returns any previously defined procedure, usually NULL.

When a GC is freed, your routine is called with these arguments:

\[(\text{proc})(\text{display}, \text{gc}, \text{codes})\]

\[
\begin{align*}
\text{Display} & \quad \text{*display}; \\
\text{GC} & \quad \text{gc}; \\
\text{XExtCodes} & \quad *\text{codes};
\end{align*}
\]

```c
int (*XESetCreateFont(display, extension, proc))()
\begin{align*}
\text{Display} & \quad *\text{display}; \\
\text{int} & \quad \text{extension}; \\
\text{int} & \quad (*\text{proc})();
\end{align*}
```

\textit{display} Specifies the connection to the X server.

\textit{extension} Specifies the extension number.

\textit{proc} Specifies the routine to call when a font is created.

You use this procedure to define a procedure to be called whenever \texttt{XLoadQueryFont} and \texttt{XQueryFont} are called. This procedure returns any previously defined procedure, usually NULL.

When \texttt{XLoadQueryFont} or \texttt{XQueryFont} is called, your routine is called with these arguments:

\[(\text{proc})(\text{display}, \text{fs}, \text{codes})\]

\[
\begin{align*}
\text{Display} & \quad *\text{display}; \\
\text{XFontStruct} & \quad *\text{fs}; \\
\text{XExtCodes} & \quad *\text{codes};
\end{align*}
\]

```c
int (*XESetFreeFont(display, extension, proc))()
\begin{align*}
\text{Display} & \quad *\text{display}; \\
\text{int} & \quad \text{extension}; \\
\text{int} & \quad (*\text{proc})();
\end{align*}
```

\textit{display} Specifies the connection to the X server.
extension  Specifies the extension number.
proc    Specifies the routine to call when a font is freed.

You use this procedure to define a procedure to be called whenever XFreeFont is called. This procedure returns any previously defined procedure, usually NULL.

When XFreeFont is called, your routine is called with these arguments:

(*proc)(display, fs, codes)
  Display *display;
  XFontStruct *fs;
  XExtCodes *codes;

The next two functions allow you to define new events to the library. An XEvent structure always has a type code (type int) as the first component. This uniquely identifies what kind of event it is. The second component is always the serial number (type unsigned long) of the last request processed by the server. The third component is always a boolean (type Bool) indicating whether the event came from a SendEvent protocol request. The fourth component is always a pointer to the display the event was read from. The fifth component is always a resource ID of one kind or another, usually a window, carefully selected to be useful to toolkit dispatchers. The fifth component should always exist, even if the event does not have a natural "destination"; if there is no value from the protocol to put in this component, initialize it to zero.

Note
There is an implementation limit such that your host event structure size cannot be bigger than the size of the XEvent union of structures. There also is no way to guarantee that more than 24 elements or 96 characters in the structure will be fully portable between machines.

int (*XESetWireToEvent(tfwp/ay, event_number, proc))()
  Display *display;
  int event_number;
  Status (*proc)();

display    Specifies the connection to the X server.
event_number Specifies the event code.
proc        Specifies the routine to call when converting an event.

You use this procedure to define a procedure to be called when an event needs to be converted from wire format (xEvent) to host format (XEvent). The event number defines which protocol event number to install a conversion routine for. This procedure returns any previously defined procedure.

Note
You can replace a core event conversion routine with one of your own, although this is not encouraged. It would, however, allow you to intercept a core event and modify it before being placed in the queue or otherwise examined.

When Xlib needs to convert an event from wire format to host format, your routine is called with these arguments:

Status (*proc)(display, re, event)
  Display *display;
  XEvent *re;
  xEvent *event;
Your routine must return status to indicate if the conversion succeeded. The re argument is a pointer to where the host format event should be stored, and the event argument is the 32-byte wire event structure. In the XEvent structure you are creating, you must fill in the five required members of the event structure. You should fill in the type member with the type specified for the xEvent structure. You should copy all other members from the xEvent structure (wire format) to the XEvent structure (host format). Your conversion routine should return True if the event should be placed in the queue or False if it should not be placed in the queue.

To initialize the serial number component of the event, call _XSetLastRequestRead with the event and use the return value.

```c
unsigned long _XSetLastRequestRead(display, rep)
    Display *display;
    xGenericReply *rep;

display    Specifies the connection to the X server.
rep        Specifies the wire event structure.
```

This function computes and returns a complete serial number from the partial serial number in the event.

```c
Status (*XESetEventToWire(display, event_number, proc))()
    Display *display;
    int event_number;
    int (*proc)();

display    Specifies the connection to the X server.
event_number Specifies the event code.
proc        Specifies the routine to call when converting an event.
```

You use this procedure to define a procedure to be called when an event needs to be converted from host format (XEvent) to wire format (xEvent) form. The event number defines which protocol event number to install a conversion routine for. This procedure returns any previously defined procedure. It returns zero if the conversion fails or nonzero otherwise.

**Note**

You can replace a core event conversion routine with one of your own, although this is not encouraged. It would, however, allow you to intercept a core event and modify it before being sent to another client.

When Xlib needs to convert an event from host format to wire format, your routine is called with these arguments:

```c
(*proc)(display, re, event)
    Display *display;
    XEvent *re;
    xEvent *event;
```

The re argument is a pointer to the host format event, and the event argument is a pointer to where the 32-byte wire event structure should be stored. You should fill in the type with the type from the XEvent structure. All other members then should be copied from the host format to the xEvent structure.
Bool (*XESetWireToError(display, error_number, proc))()

Display *display;
int error_number;
Bool (*proc)();

display Specifies the connection to the X server.
error_number Specifies the error code.
proc Specifies the routine to call when an error is received.

This function defines a procedure to be called when an extension error needs to be converted from wire format to host format. The error number defines which protocol error code to install the conversion routine for. This procedure returns any previously defined procedure.

Use this function for extension errors that contain additional error values beyond those in a core X error, when multiple wire errors must be combined into a single Xlib error, or when it is necessary to intercept an X error before it is otherwise examined.

When Xlib needs to convert an error from wire format to host format, the routine is called with these arguments:

Bool (*proc)(display, he, we)

Display *display;
XErrorEvent *he;
xError *we;

The he argument is a pointer to where the host format error should be stored. The structure pointed at by he is guaranteed to be as large as an XEvent structure, and so can be cast to a type larger than an XErrorEvent, in order to store additional values. If the error is to be completely ignored by Xlib (for example, several protocol error structures will be combined into one Xlib error), then the function should return False; otherwise it should return True.

int (*XESetError(display, extension, proc))()

Display *display;
int extension;
int (*proc)();

display Specifies the connection to the X server.
extension Specifies the extension number.
proc Specifies the routine to call when an error is received.

Inside Xlib, there are times that you may want to suppress the calling of the external error handling when an error occurs. This allows status to be returned on a call at the cost of the call being synchronous (though most such routines are query operations, in any case, and are typically programmed to be synchronous).

When Xlib detects a protocol error in _XReply, it calls your procedure with these arguments:

int (*proc)(display, err, codes, ret_code)

Display *display;
xError *err;
XExtCodes *codes;
int *ret_code;

The err argument is a pointer to the 32-byte wire format error. The codes argument is a pointer to the extension codes structure. The ret_code argument is the return code you may want _XReply returned to.

If your routine returns a zero value, the error is not suppressed, and the client's error handler is called. (For further information, see section 11.8.2.) If your routine returns nonzero, the error is suppressed, and _XReply returns the value of ret_code.
char *(*XESetErrorString(display, extension, proc))()
  Display *display;
  int extension;
  char *(*proc)();

display Specifies the connection to the X server.
extension Specifies the extension number.
proc specifies the routine to call to obtain an error string.

The XGetErrorText function returns a string to the user for an error. XESetErrorString allows you to define a routine to be called that should return a pointer to the error message. The following is an example.

(*proc)(display, code, codes, buffer, nbytes)
  Display *display;
  int code;
  XExtCodes *codes;
  char *buffer;
  int nbytes;

Your procedure is called with the error code for every error detected. You should copy nbytes of a null-terminated string containing the error message into buffer.

void (*XESetPrintErrorValues(display, extension, proc))()
  Display *display;
  int extension;
  void (*proc)();

display Specifies the connection to the X server.
extension Specifies the extension number.
proc Specifies the routine to call when an error is printed.

This function defines a procedure to be called when an extension error is printed, to print the error values. Use this function for extension errors that contain additional error values beyond those in a core X error. This function returns any previously defined procedure.

When Xlib needs to print an error, the routine is called with these arguments:

void (*proc)(display, ev, fp)
  Display *display;
  XErrorEvent *ev;
  void *fp;

The structure pointed at by ev is guaranteed to be as large as an XEvent structure, and so can be cast to a type larger than an XErrorEvent, in order to obtain additional values set by using XESetWireToError. The underlying type of the fp argument is system dependent; on a POSIX-compliant fp should be cast to type FILE*.

int (*XESetFlushGC(display, extension, proc))()
  Display *display;
  int extension;
  int *(*proc)();

display Specifies the connection to the X server.
extension Specifies the extension number.
proc Specifies the routine to call when a GC is flushed.

The XESetFlushGC procedure is identical to XESetCopyGC except that XESetFlushGC is called when a GC cache needs to be updated in the server.
Hooks onto Xlib Data Structures

Various Xlib data structures have provisions for extension routines to chain extension supplied data onto a list. These structures are GC, Visual, Screen, ScreenFormat, Display, and XFontStruct. Because the list pointer is always the first member in the structure, a single set of routines can be used to manipulate the data on these lists.

The following structure is used in the routines in this section and is defined in `<X11/Xlib.h>`:

```c
typedef struct _XExtData {
    int number; /* number returned by XInitExtension */
    struct _XExtData *next; /* next item on list of data for structure */
    int (*free_private)(0); /* if defined, called to free private */
    XPointer private_data; /* data private to this extension. */
} XExtData;
```

When any of the data structures listed above are freed, the list is walked, and the structure’s free routine (if any) is called. If free is NULL, then the library frees both the data pointed to by the private_data member and the structure itself.

```c
union { Display *display;
    GC gc;
    Visual *visual;
    Screen *screen;
    ScreenFormat *pixmap_format;
    XFontStruct *font } XEDataObject;
```

```c
XExtData **XEHeadOfExtensionList(object)
    XEDataObject object;

object Specifies the object.

XEHeadOfExtensionList returns a pointer to the list of extension structures attached to the specified object. In concert with XAddToExtensionList, XEHeadOfExtensionList allows an extension to attach arbitrary data to any of the structures of types contained in XEDataObject.

```c
XAddToExtensionList(structure, ext_data)
    XExtData **structure;
    XExtData *ext_data;

structure Specifies the extension list.

ext_data Specifies the extension data structure to add.
```

The structure argument is a pointer to one of the data structures enumerated above. You must initialize ext_data->number with the extension number before calling this routine.

```c
XExtData *XFindOnExtensionList(structure, number)
    struct _XExtData **structure;
    int number;

structure Specifies the extension list.

number Specifies the extension number from XInitExtension.
```

XFindOnExtensionList returns the first extension data structure for the extension numbered number. It is expected that an extension will add at most one extension data structure to any single data structure’s extension data list. There is no way to find additional structures.

The XAllocID macro, which allocates and returns a resource ID, is defined in `<X11/Xlib.h>`.

```c
XAllocID(display)
    Display *display;
```
display Specifies the connection to the X server.
This macro is a call through the Display structure to the internal resource ID allocator. It returns a resource ID that you can use when creating new resources.

GC Caching
GCs are cached by the library to allow merging of independent change requests to the same GC into single protocol requests. This is typically called a write-back cache. Any extension routine whose behavior depends on the contents of a GC must flush the GC cache to make sure the server has up-to-date contents in its GC.
The FlushGC macro checks the dirty bits in the library’s GC structure and calls _XFlushGCCache if any elements have changed. The FlushGC macro is defined as follows:

FlushGC(display, gc)
    Display *display;
    GC gc;

display Specifies the connection to the X server.
gc Specifies the GC.
Note that if you extend the GC to add additional resource ID components, you should ensure that the library stub sends the change request immediately. This is because a client can free a resource immediately after using it, so if you only stored the value in the cache without forcing a protocol request, the resource might be destroyed before being set into the GC. You can use the _XFlushGCCache procedure to force the cache to be flushed. The _XFlushGCCache procedure is defined as follows:

_XFlushGCCache(display, gc)
    Display *display;
    GC gc;

display Specifies the connection to the X server.
gc Specifies the GC.

Graphics Batching
If you extend X to add more poly graphics primitives, you may be able to take advantage of facilities in the library to allow back-to-back single calls to be transformed into poly requests. This may dramatically improve performance of programs that are not written using poly requests. A pointer to an xReq, called last_req in the display structure, is the last request being processed. By checking that the last request type, drawable, gc, and other options are the same as the new one and that there is enough space left in the buffer, you may be able to just extend the previous graphics request by extending the length field of the request and appending the data to the buffer. This can improve performance by five times or more in naive programs. For example, here is the source for the XDrawPoint stub. (Writing extension stubs is discussed in the next section.)

#include "copyright.h"

#include "Xlibint.h"

/* precompute the maximum size of batching request allowed */
static int size = sizeof(xPolyPointReq) + EPERBATCH * sizeof(xPoint);

XDrawPoint(dpy, d, gc, x, y)
    register Display *dpy;
    Drawable d;
To keep clients from generating very long requests that may monopolize the server, there is a
symbol defined in <X11/Xlibint.h> of EPERBATCH on the number of requests batched.
Most of the performance benefit occurs in the first few merged requests. Note that FlushGC
is called before picking up the value of last_req, because it may modify this field.

Writing Extension Stubs

All X requests always contain the length of the request, expressed as a 16-bit quantity of 32
bits. This means that a single request can be no more than 256K bytes in length. Some
servers may not support single requests of such a length. The value of dpy->max_request_size
contains the maximum length as defined by the server implementation. For further informa-
tion, see "X Window System Protocol."

Requests, Replies, and Xproto.h

The <X11/Xproto.h> file contains three sets of definitions that are of interest to the stub
implementor: request names, request structures, and reply structures.

You need to generate a file equivalent to <X11/Xproto.h> for your extension and need to
include it in your stub routine. Each stub routine also must include <X11/Xlibint.h>.

The identifiers are deliberately chosen in such a way that, if the request is called
X_DoSomething, then its request structure is xDoSomethingReq, and its reply is
xDoSomethingReply. The GetReq family of macros, defined in <X11/Xlibint.h>, takes advantage of this naming scheme.

For each X request, there is a definition in <X11/Xproto.h> that looks similar to this:

```
#define X_DoSomething 42
```

In your extension header file, this will be a minor opcode, instead of a major opcode.

**Request Format**

Every request contains an 8-bit major opcode and a 16-bit length field expressed in units of four bytes. Every request consists of four bytes of header (containing the major opcode, the length field, and a data byte) followed by zero or more additional bytes of data. The length field defines the total length of the request, including the header. The length field in a request must equal the minimum length required to contain the request. If the specified length is smaller or larger than the required length, the server should generate a **BadLength** error.

Unused bytes in a request are not required to be zero. Extensions should be designed in such a way that long protocol requests can be split up into smaller requests, if it is possible to exceed the maximum request size of the server. The protocol guarantees the maximum request size to be no smaller than 4096 units (16384 bytes).

Major opcodes 128 through 255 are reserved for extensions. Extensions are intended to contain multiple requests, so extension requests typically have an additional minor opcode encoded in the “spare” data byte in the request header, but the placement and interpretation of this minor opcode as well as all other fields in extension requests are not defined by the core protocol. Every request is implicitly assigned a sequence number (starting with one) used in replies, errors, and events.

To help but not cure portability problems to certain machines, the B16 and B32 macros have been defined so that they can become bitfield specifications on some machines. For example, on a Cray, these should be used for all 16-bit and 32-bit quantities, as discussed below.

Most protocol requests have a corresponding structure typedef in <X11/Xproto.h>, which looks like:

```
typedef struct _DoSomethingReq {
    CARD8 reqType; /* X_DoSomething */
    CARD8 someDatum; /* used differently in different requests */
    CARD 16 length B16; /* total # of bytes in request, divided by 4 */
    ...
    /* request-specific data */
    ...
} xDoSomethingReq;
```

If a core protocol request has a single 32-bit argument, you need not declare a request structure in your extension header file. Instead, such requests use <X11/Xproto.h>’s xResourceReq structure. This structure is used for any request whose single argument is a Window, Pixmap, Drawable, GContext, Font, Cursor, Colormap, Atom, or VisualID.

```
typedef struct _ResourceReq {
    CARD8 reqType; /* the request type, e.g. X_DoSomething */
    BYTE pad; /* not used */
    CARD16 length B16; /* 2 (= total # of bytes in request, divided by 4) */
    CARD32 id B32; /* the Window, Drawable, Font, GContext, etc. */
} xResourceReq;
```

If convenient, you can do something similar in your extension header file.

In both of these structures, the reqType field identifies the type of the request (for example, X_MapWindow or X_CreatePixmap). The length field tells how long the request is in units of
4-byte longwords. This length includes both the request structure itself and any variable length data, such as strings or lists, that follow the request structure. Request structures come in different sizes, but all requests are padded to be multiples of four bytes long.

A few protocol requests take no arguments at all. Instead, they use `<X11/Xproto.h>`’s `xReq` structure, which contains only a reqType and a length (and a pad byte).

If the protocol request requires a reply, then `<X11/Xproto.h>` also contains a reply structure typedef:

```c
typedef struct _DoSomethingReply {
    BYTE type; /* always X_Reply */
    BYTE someDatum; /* used differently in different requests */
    CARD16 sequenceNumber B16; /* # of requests sent so far */
    CARD32 length B32; /* # of additional bytes, divided by 4 */
    ... /* request-specific data */
    ... } xDoSomethingReply;
```

Most of these reply structures are 32 bytes long. If there are not that many reply values, then they contain a sufficient number of pad fields to bring them up to 32 bytes. The length field is the total number of bytes in the request minus 32, divided by 4. This length will be nonzero only if:

- The reply structure is followed by variable length data such as a list or string.
- The reply structure is longer than 32 bytes.

Only `GetWindowAttributes`, `QueryFont`, `QueryKeymap`, and `GetKeyboardControl` have reply structures longer than 32 bytes in the core protocol.

A few protocol requests return replies that contain no data. `<X11/Xproto.h>` does not define reply structures for these. Instead, they use the `xGenericReply` structure, which contains only a type, length, and sequence number (and sufficient padding to make it 32 bytes long).

### Starting to Write a Stub Routine

An Xlib stub routine should always start like this:

```c
#include "Xlibint.h"

XDoSomething (arguments, ... )
/* argument declarations */
{

    register XDoSomethingReq *req;

```

If the protocol request has a reply, then the variable declarations should include the reply structure for the request. The following is an example:

```c
    xDoSomethingReply rep;
```

### Locking Data Structures

To lock the display structure for systems that want to support multithreaded access to a single display connection, each stub will need to lock its critical section. Generally, this section is the point from just before the appropriate GetReq call until all arguments to the call have been stored into the buffer. The precise instructions needed for this locking depend upon the machine architecture. Two calls, which are generally implemented as macros, have been provided.
LockDisplay(display)
    Display *display;

UnlockDisplay(display)
    Display *display;

display Specifies the connection to the X server.

Sending the Protocol Request and Arguments
After the variable declarations, a stub routine should call one of four macros defined in
<Xlib/Xlibint.h>: GetReq, GetReqExtra, GetResReq, or GetEmptyReq. All of these
macros take, as their first argument, the name of the protocol request as declared in
<Xlib/Xproto.h> except with X_ removed. Each one declares a Display structure pointer,
called dpy, and a pointer to a request structure, called req, which is of the appropriate type.
The macro then appends the request structure to the output buffer, fills in its type and length
field, and sets req to point to it.
If the protocol request has no arguments (for instance, X_GrabServer), then use GetEmptyReq.

    GetEmptyReq (DoSomething, req);

If the protocol request has a single 32-bit argument (such as a Pixmap, Window, Drawable,
Atom, and so on), then use GetResReq. The second argument to the macro is the 32-bit
object. X_MapWindow is a good example.

    GetResReq (DoSomething, rid, req);

The rid argument is the Pixmap, Window, or other resource ID.
If the protocol request takes any other argument list, then call GetReq. After the GetReq,
you need to set all the other fields in the request structure, usually from arguments to the stub
routine.

    GetReq (DoSomething, req);
    /* fill in arguments here */
    req->arg1 = arg1;
    req->arg2 = arg2;

A few stub routines (such as XCreateGC and XCreatePixmap) return a resource ID to the
caller but pass a resource ID as an argument to the protocol request. Such routines use the
macro XAllocID to allocate a resource ID from the range of IDs that were assigned to this
client when it opened the connection.

    rid = req->rid = XAllocID();
    return (rid);

Finally, some stub routines transmit a fixed amount of variable length data after the request.
Typically, these routines (such as XMoveWindow and XSetBackground) are special cases of
more general functions like XMoveResizeWindow and XChangeGC. These special case
routines use GetReqExtra, which is the same as GetReq except that it takes an additional
argument (the number of extra bytes to allocate in the output buffer after the request structure).
This number should always be a multiple of four.

Variable Length Arguments
Some protocol requests take additional variable length data that follow the xDoSomethingReq
structure. The format of this data varies from request to request. Some requests require a
sequence of 8-bit bytes, others a sequence of 16-bit or 32-bit entities, and still others a
sequence of structures.
It is necessary to add the length of any variable length data to the length field of the request structure. That length field is in units of 32-bit longwords. If the data is a string or other sequence of 8-bit bytes, then you must round the length up and shift it before adding:

\[
\text{req->length} += (\text{nbytes}+3)>>2;
\]

To transmit variable length data, use the Data macros. If the data fits into the output buffer, then this macro copies it to the buffer. If it does not fit, however, the Data macro calls _XSend, which transmits first the contents of the buffer and then your data. The Data macros take three arguments: the Display, a pointer to the beginning of the data, and the number of bytes to be sent.

\[
\text{Data(display, (char *) data, nbytes)};
\]

\[
\text{Data16(display, (short *) data, nbytes)};
\]

\[
\text{Data32(display, (long *) data, nbytes)};
\]

Data, Data16, and Data32 are macros that may use their last argument more than once, so that argument should be a variable rather than an expression such as “nitems*sizeof(item)”. You should do that kind of computation in a separate statement before calling them. Use the appropriate macro when sending byte, short, or long data.

If the protocol request requires a reply, then call the procedure _XSend instead of the Data macro. _XSend takes the same arguments, but because it sends your data immediately instead of copying it into the output buffer (which would later be flushed anyway by the following call on _XReply), it is faster.

Replies

If the protocol request has a reply, then call _XReply after you have finished dealing with all the fixed and variable length arguments. _XReply flushes the output buffer and waits for an xReply packet to arrive. If any events arrive in the meantime, _XReply places them in the queue for later use.

\[
\text{Status _XReply(display, rep, extra, discard)}
\]

\[
\text{Display *display};
\]

\[
\text{xReply *rep};
\]

\[
\text{int extra};
\]

\[
\text{Bool discard};
\]

\[
\text{display} \quad \text{Specifies the connection to the X server.}
\]

\[
\text{rep} \quad \text{Specifies the reply structure.}
\]

\[
\text{extra} \quad \text{Specifies the number of 32-bit words expected after the reply.}
\]

\[
\text{discard} \quad \text{Specifies if beyond the "extra" data should be discarded.}
\]

_XReply waits for a reply packet and copies its contents into the specified rep. _XReply handles error and event packets that occur before the reply is received. _XReply takes four arguments:

- A Display * structure
- A pointer to a reply structure (which must be cast to an xReply *)
- The number of additional 32-bit words (beyond sizeof(xReply) = 32 bytes) in the reply structure
- A Boolean that indicates whether _XReply is to discard any additional bytes beyond those it was told to read

Because most reply structures are 32 bytes long, the third argument is usually 0. The only core protocol exceptions are the replies to GetWindowAttributes, QueryFont, QueryKeymap,
and `GetKeyboardControl`, which have longer replies.

The last argument should be `False` if the reply structure is followed by additional variable length data (such as a list or string). It should be `True` if there is not any variable length data.

Note

This last argument is provided for upward-compatibility reasons to allow a client to communicate properly with a hypothetical later version of the server that sends more data than the client expected. For example, some later version of `GetWindowAttributes` might use a larger, but compatible, `xGetWindowAttributesReply` that contains additional attribute data at the end.

`_XReply` returns `True` if it received a reply successfully or `False` if it received any sort of error.

For a request with a reply that is not followed by variable length data, you write something like:

```c
_XReply(display, (xReply *)&rep, 0, True);
*ret1 = rep.ret1;
*ret2 = rep.ret2;
*ret3 = rep.ret3;
UnlockDisplay(dpy);
SyncHandleO;
return (rep.ret4);
}
```

If there is variable length data after the reply, change the `True` to `False`, and use the appropriate `_XRead` function to read the variable length data.

```c
_XRead(display, data_return, nbytes)
Display *display;
char *data_return;
long nbytes;
```

Specifies the connection to the X server.

Specifies the buffer.

Specifies the number of bytes required.

`_XRead` reads the specified number of bytes into `data_return`.

```c
_XRead16(display, data_return, nbytes)
Display *display;
short *data_return;
long nbytes;
```

Specifies the connection to the X server.

Specifies the buffer.

Specifies the number of bytes required.

`_XRead16` reads the specified number of bytes, unpacking them as 16-bit quantities, into the specified array as shorts.

```c
_XRead32(display, data_return, nbytes)
Display *display;
long *data_return;
long nbytes;
```

Specifies the connection to the X server.

Specifies the buffer.

Specifies the number of bytes required.

`_XRead32` reads the specified number of bytes, unpacking them as 32-bit quantities, into the specified array as longs.
display Specifies the connection to the X server.
data_return Specifies the buffer.
nbytes Specifies the number of bytes required.

_XRead32 reads the specified number of bytes, unpacking them as 32-bit quantities, into the specified array as longs.

_XRead16Pad(display, data_return, nbytes)
Display *display;
short *data_return;
long nbytes;

display Specifies the connection to the X server.
data_return Specifies the buffer.
nbytes Specifies the number of bytes required.

_XRead16Pad reads the specified number of bytes, unpacking them as 16-bit quantities, into the specified array as shorts. If the number of bytes is not a multiple of four, _XRead16Pad reads and discards up to three additional pad bytes.

_XReadPad(display, data_return, nbytes)
Display *display;
char *data_return;
long nbytes;

display Specifies the connection to the X server.
data_return Specifies the buffer.
nbytes Specifies the number of bytes required.

_XReadPad reads the specified number of bytes into data_return. If the number of bytes is not a multiple of four, _XReadPad reads and discards up to three additional pad bytes.

Each protocol request is a little different. For further information, see the Xlib sources for examples.

Synchronous Calling
To ease debugging, each routine should have a call, just before returning to the user, to a routine called SyncHandle. This routine generally is implemented as a macro. If synchronous mode is enabled (see XSynchronize), the request is sent immediately. The library, however, waits until any error the routine could generate at the server has been handled.

Allocating and Deallocating Memory
To support the possible reentry of these routines, you must observe several conventions when allocating and deallocating memory, most often done when returning data to the user from the window system of a size the caller could not know in advance (for example, a list of fonts or a list of extensions). The standard C library routines on many systems are not protected against signals or other multithreaded uses. The following analogies to standard I/O library routines have been defined:

Xmalloc() Replaces malloc()
XFree() Replaces free()
Xcalloc() Replaces calloc()

These should be used in place of any calls you would make to the normal C library routines.

If you need a single scratch buffer inside a critical section (for example, to pack and unpack data to and from the wire protocol),
the general memory allocators may be too expensive to use (particularly in output routines,
which are performance critical). The routine below returns a scratch buffer for your use:

```c
char * XAllocScratch(display, nbytes)
    Display *display;
    unsigned long nbytes;

display  Specifies the connection to the X server.
 nbytes  Specifies the number of bytes required.
```

This storage must only be used inside of the critical section of your stub.

Portability Considerations

Many machine architectures, including many of the more recent RISC architectures, do not correctly access data at unaligned locations; their compilers pad out structures to preserve this characteristic. Many other machines capable of unaligned references pad inside of structures as well to preserve alignment, because accessing aligned data is usually much faster. Because the library and the server use structures to access data at arbitrary points in a byte stream, all data in request and reply packets must be naturally aligned; that is, 16-bit data starts on 16-bit boundaries in the request and 32-bit data on 32-bit boundaries. All requests must be a multiple of 32 bits in length to preserve the natural alignment in the data stream. You must pad structures out to 32-bit boundaries. Pad information does not have to be zeroed unless you want to preserve such fields for future use in your protocol requests. Floating point varies radically between machines and should be avoided completely if at all possible.

This code may run on machines with 16-bit ints. So, if any integer argument, variable, or return value either can take only nonnegative values or is declared as a CARD16 in the protocol, be sure to declare it as unsigned int and not as int. (This, of course, does not apply to Booleans or enumerations.)

Similarly, if any integer argument or return value is declared CARD32 in the protocol, declare it as an unsigned long and not as int or long. This also goes for any internal variables that may take on values larger than the maximum 16-bit unsigned int.

The library currently assumes that a char is 8 bits, a short is 16 bits, an int is 16 or 32 bits, and a long is 32 bits. The PackData macro is a half-hearted attempt to deal with the possibility of 32 bit shorts. However, much more work is needed to make this work properly.

Deriving the Correct Extension Opcode

The remaining problem a writer of an extension stub routine faces that the core protocol does not face is to map from the call to the proper major and minor opcodes. While there are a number of strategies, the simplest and fastest is outlined below.

1. Declare an array of pointers, _NFILE long (this is normally found in <stdio.h> and is the number of file descriptors supported on the system) of type XExtCodes. Make sure these are all initialized to NULL.

2. When your stub is entered, your initialization test is just to use the display pointer passed in to access the file descriptor and an index into the array. If the entry is NULL, then this is the first time you are entering the routine for this display. Call your initialization routine and pass it to the display pointer.

3. Once in your initialization routine, call XInitExtension; if it succeeds, store the pointer returned into this array. Make sure to establish a close display handler to allow you to zero the entry. Do whatever other initialization your extension requires. (For example, install event handlers and so on.) Your initialization routine would normally return a pointer to the XExtCodes structure for this extension, which is what would normally be found in your array of pointers.

4. After returning from your initialization routine, the stub can now continue normally, because it has its major opcode safely in its hand in the XExtCodes structure.
Appendix D
Compatibility Functions

The X Version 11 and X Version 10 functions discussed in this appendix are obsolete, have been superseded by newer X Version 11 functions, and are maintained for compatibility reasons only.

X Version 11 Compatibility Functions
You can use the X Version 11 compatibility functions to:

- Set standard properties
- Set and get window sizing hints
- Set and get an XStandardColormap structure
- Parse window geometry
- Get X environment defaults

Setting Standard Properties
To specify a minimum set of properties describing the "quickie" application, use XSetStandardProperties. This function has been superseded by XSetWMProperties and sets all or portions of the WM_NAME, WM_ICON_NAME, WM_HINTS, WM_COMMAND, and WM_NORMAL_HINTS properties.

XSetStandardProperties(display, w, window_name, icon_name, icon_pixmap, argv, argc, hints)

Display *display;
Window w;
char *window_name;
char *icon_name;
Pixmap icon_pixmap;
char **argv;
int argc;
XSizeHints *hints;

display Specifies the connection to the X server.
w Specifies the window.
window_name Specifies the window name, which should be a null-terminated string.
icon_name Specifies the icon name, which should be a null-terminated string.
icon_pixmap Specifies the bitmap that is to be used for the icon or None.
argv Specifies the application's argument list.
argc Specifies the number of arguments.
hints Specifies a pointer to the size hints for the window in its normal state.

The XSetStandardProperties function provides a means by which simple applications set the most essential properties with a single call. XSetStandardProperties should be used to give a window manager some information about your program's preferences. It should not be used by applications that need to communicate more information than is possible with XSetStandardProperties. (Typically, argv is the argv array of your main program.) If the strings are not in the Host Portable Character Encoding the result is implementation dependent.

354
XSetStandardProperties can generate BadAlloc and BadWindow errors.

Setting and Getting Window Sizing Hints

Xlib provides functions that you can use to set or get window sizing hints. The functions discussed in this section use the flags and the XSizeHints structure, as defined in the <X11/Xutil.h> header file, and use the WM_NORMAL_HINTS property.

To set the size hints for a given window in its normal state, use XSetNormalHints. This function has been superseded by XSetWMNormalHints.

XSetNormalHints(display, w, hints)

Display *display;
Window w;
XSizeHints *hints;

display Specifies the connection to the X server.
w Specifies the window.
hints Specifies a pointer to the size hints for the window in its normal state.

The XSetNormalHints function sets the size hints structure for the specified window. Applications use XSetNormalHints to inform the window manager of the size or position desirable for that window. In addition, an application that wants to move or resize itself should call XSetNormalHints and specify its new desired location and size as well as making direct Xlib calls to move or resize. This is because window managers may ignore redirected configure requests, but they pay attention to property changes.

To set size hints, an application not only must assign values to the appropriate members in the hints structure but also must set the flags member of the structure to indicate which information is present and where it came from. A call to XSetNormalHints is meaningless, unless the flags member is set to indicate which members of the structure have been assigned values.

XSetNormalHints can generate BadAlloc and BadWindow errors.

To return the size hints for a window in its normal state, use XGetNormalHints. This function has been superseded by XGetWMNormalHints.

Status XGetNormalHints(display, w, hints)_return

Display *display;
Window w;
XSizeHints *hints_return;

display Specifies the connection to the X server.
w Specifies the window.
hints_return Returns the size hints for the window in its normal state.

The XGetNormalHints function returns the size hints for a window in its normal state. It returns a nonzero status if it succeeds or zero if the application specified no normal size hints for this window.

XGetNormalHints can generate a BadWindow error.

The next two functions set and read the WM_ZOOM_HINTS property.

To set the zoom hints for a window, use XSetZoomHints. This function is no longer supported by the Inter-Client Communication Conventions Manual.
XSetZoomHints(display, w, zhints)
  Display *display;
  Window w;
  XSizeHints *zhints;

display     Specifies the connection to the X server.
w          Specifies the window.
zhints    Specifies a pointer to the zoom hints.

Many window managers think of windows in one of three states: iconic, normal, or zoomed. The XSetZoomHints function provides the window manager with information for the window in the zoomed state.

XSetZoomHints can generate BadAlloc and BadWindow errors.

To read the zoom hints for a window, use XGetZoomHints. This function is no longer supported by the Inter-Client Communication Conventions Manual.

Status XGetZoomHints(display, w, zhints_return)
  Display *display;
  Window w;
  XSizeHints *zhints_return;

display     Specifies the connection to the X server.
w          Specifies the window.
zhints_return Returns the zoom hints.

The XGetZoomHints function returns the size hints for a window in its zoomed state. It returns a nonzero status if it succeeds or zero if the application specified no zoom size hints for this window.

XGetZoomHints can generate a BadWindow error.

To set the value of any property of type WM_SIZE_HINTS, use XSetSizeHints. This function has been superseded by XSetWMSizeHints.

XSetSizeHints(display, w, hints, property)
  Display *display;
  Window w;
  XSizeHints *hints;
  Atom property;

display     Specifies the connection to the X server.
w          Specifies the window.
hints    Specifies a pointer to the size hints.
property    Specifies the property name.

The XSetSizeHints function sets the XSizeHints structure for the named property and the specified window. This is used by XSetNormalHints and XSetZoomHints, and can be used to set the value of any property of type WM_SIZE_HINTS. Thus, it may be useful if other properties of that type get defined.

XSetSizeHints can generate BadAlloc, BadAtom, and BadWindow errors.

To read the value of any property of type WM_SIZE_HINTS, use XGetSizeHints. This function has been superseded by XGetWMSizeHints.
Status XGetSizeHints(display, w, hints_return, property)
Monitor *display;
Window w;
XSizeHints *hints_return;
Atom property;

display Specifies the connection to the X server.
w Specifies the window.
hints_return Returns the size hints.
property Specifies the property name.

XGetSizeHints returns the XSizeHints structure for the named property and the specified window. This is used by XGetNormalHints and XGetZoomHints. It also can be used to retrieve the value of any property of type WM_SIZE_HINTS. Thus, it may be useful if other properties of that type get defined. XGetSizeHints returns a nonzero status if a size hint was defined or zero otherwise.

XGetSizeHints can generate BadAtom and BadWindow errors.

Getting and Setting an XStandardColormap Structure

To get the XStandardColormap structure associated with one of the described atoms, use XGetStandardColormap. This function has been superseded by XGetRGBColormap.

Status XGetStandardColormap(display, w, colormap_return, property)
Monitor *display;
Window w;
XStandardColormap *colormap_return;
Atom property;

/* RGB_BEST_MAP, etc. */
display Specifies the connection to the X server.
w Specifies the window.
colormap_return Returns the colormap associated with the specified atom.
property Specifies the property name.

The XGetStandardColormap function returns the colormap definition associated with the atom supplied as the property argument. XGetStandardColormap returns a nonzero status if successful, and zero otherwise. For example, to fetch the standard Grayscale colormap for a display, you use XGetStandardColormap with the following syntax:

XGetStandardColormap(dpy, DefaultRootWindow(dpy), &cmap, XA_RGB_GRAY_MAP);

See section 14.3 for the semantics of standard colormaps.

XGetStandardColormap can generate BadAtom and BadWindow errors.

To set a standard colormap, use XSetStandardColormap. This function has been superseded by XSetRGBColormap.

XSetStandardColormap(display, w, colormap, property)
Monitor *display;
Window w;
XStandardColormap *colormap;
Atom property;

/* RGB_BEST_MAP, etc. */
display Specifies the connection to the X server.
w Specifies the window.
colormap Specifies the colormap.
property

Specifies the property name.

The XSetStandardColormap function usually is only used by window or session managers. XSetStandardColormap can generate BadAlloc, BadAtom, BadDrawable, and BadWindow errors.

Parsing Window Geometry

To parse window geometry given a user-specified position and a default position, use XGeometry. This function has been superseded by XWMGeometry.

```c
int XGeometry(display, screen, position, default_position, bwidth, fwidth, fheight, xadder, yadder, x_return, y_return, width_return, height_return)
```

- **display** Specifies the connection to the X server.
- **screen** Specifies the screen.
- **position** Specify the geometry specifications.
- **default_position** Specify the geometry specifications.
- **bwidth** Specifies the border width.
- **fwidth** Specify the font height and width in pixels (increment size).
- **xadder** Specify additional interior padding needed in the window.
- **yadder** Specify additional interior padding needed in the window.
- **x_return** Return the x offset.
- **y_return** Return the y offset.
- **width_return** Return the width determined.
- **height_return** Return the height determined.

You pass in the border width (bwidth), size of the increments fwidth and fheight (typically font width and height), and any additional interior space (xadder and yadder) to make it easy to compute the resulting size. The XGeometry function returns the position the window should be placed given a position and a default position. XGeometry determines the placement of a window using a geometry specification as specified by XParseGeometry and the additional information about the window. Given a fully qualified default geometry specification and an incomplete geometry specification, XParseGeometry returns a bitmask value as defined above in the XParseGeometry call, by using the position argument.

The returned width and height will be the width and height specified by default_position as overridden by any user-specified position. They are not affected by fwidth, fheight, xadder, or yadder. The x and y coordinates are computed by using the border width, the screen width and height, padding as specified by xadder and yadder, and the fheight and fwidth times the width and height from the geometry specifications.

Obtaining the X Environment Defaults

The XGetDefault function provides a primitive interface to the resource manager facilities discussed in chapter 15. It is only useful in very simple applications.
char *XGetDefault(display, program, option)
    Display *display;
    char *program;
    char *option;

display Specifies the connection to the X server.
program Specifies the program name for the Xlib defaults (usually argv[0] of the main
    program).
option Specifies the option name.

The XGetDefault function returns the value of the resource prog.option, where prog is the
    program argument with the directory prefix removed and option must be a single component.
Note that multi-level resources cannot be used with XGetDefault. The class "Program.Name"
    is always used for the resource lookup. If the specified option name does not exist for this
    program, XGetDefault returns NULL. The strings returned by XGetDefault are owned by
    Xlib and should not be modified or freed by the client.

If a database has been set with XrmSetDatabase, that database is used for the lookup. Other¬
    wise, a database is created as described below, and this is set in the display (as if by calling
    XrmSetDatabase). The database is created in the current locale. To create a database, XGet¬
    Default uses resources from the RESOURCE_MANAGER property on the root window of
    screen zero. If no such property exists, a resource file in the user’s home directory is used.
On a POSIX-conformant system, this file is $HOME/.Xdefaults. After loading these defaults,
XGetDefault merges additional defaults specified by the XENVIRONMENT environment
    variable. If XENVIRONMENT is defined, it contains a full path name for the additional
resource file. If XENVIRONMENT is not defined, XGetDefault looks for
    $HOME/.Xdefaults-name, where name specifies the name of the machine on which the appli¬
    cation is running.

X Version 10 Compatibility Functions
You can use the X Version 10 compatibility functions to:
• Draw and fill polygons and curves
• Associate user data with a value

Drawing and Filling Polygons and Curves
Xlib provides functions that you can use to draw or fill arbitrary polygons or curves. These
    functions are provided mainly for compatibility with X Version 10 and have no server support.
That is, they call other Xlib functions, not the server directly. Thus, if you just have straight
    lines to draw, using XDrawLines or XDrawSegments is much faster.

The functions discussed here provide all the functionality of the X Version 10 functions
    XDraw, XDrawFilled, XDrawPatterned, XDrawDashed, and XDrawTiled. They are as
compatible as possible given X Version 11's new line drawing functions. One thing to note,
    however, is that VertexDrawLastPoint is no longer supported. Also, the error status returned
is the opposite of what it was under X Version 10 (this is the X Version 11 standard error
    status). XAppendVertex and XClearVertexFlag from X Version 10 also are not supported.

Just how the graphics context you use is set up actually determines whether you get dashes or
not, and so on. Lines are properly joined if they connect and include the closing of a closed
    figure (see XDrawLines). The functions discussed here fail (return zero) only if they run out
of memory or are passed a Vertex list that has a Vertex with VertexStartClosed set that is
    not followed by a Vertex with VertexEndClosed set.

To achieve the effects of the X Version 10 XDraw, XDrawDashed, and XDrawPatterned,
use XDraw.
#include <X11/X10.h>

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

display     Specifies the connection to the X server.
d       Specifies the drawable.
gc       Specifies the GC.
vlist     Specifies a pointer to the list of vertices that indicate what to draw.
vcount    Specifies how many vertices are in vlist.

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

Each Vertex, as defined in <X11/X10.h>, is a structure with the following members:

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

The x and y members are the coordinates of the vertex that are relative to either the upper-left inside corner of the drawable (if VertexRelative is zero) or the previous vertex (if VertexRelative is one).

The flags, as defined in <X11/X10.h>, are as follows:

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

- If VertexRelative is not set, the coordinates are absolute (that is, relative to the drawable’s origin). The first vertex must be an absolute vertex.
- If VertexDontDraw is one, no line or curve is drawn from the previous vertex to this one. This is analogous to picking up the pen and moving to another place before drawing another line.
- If VertexCurved is one, a spline algorithm is used to draw a smooth curve from the previous vertex through this one to the next vertex. Otherwise, a straight line is drawn from the previous vertex to this one. It makes sense to set VertexCurved to one only if a previous and next vertex are both defined (either explicitly in the array or through the definition of a closed curve).
- It is permissible for VertexDontDraw bits and VertexCurved bits both to be one. This is useful if you want to define the previous point for the smooth curve but do not want an actual curve drawing to start until this point.
- If VertexStartClosed is one, then this point marks the beginning of a closed curve. This vertex must be followed later in the array by another vertex whose effective coordinates are identical and that has a VertexEndClosed bit of one. The points in between form a cycle to determine predecessor and successor vertices for the spline algorithm.
This function uses these GC components: function, plane-mask, line-width, line-style, cap-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. It also uses these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

To achieve the effects of the X Version 10 XDrawTiled and XDrawFilled, use XDrawFilled.

#include <X11/X10.h>

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

display Specifies the connection to the X server.
d Specifies the drawable.
gc Specifies the GC.
vlist Specifies a pointer to the list of vertices that indicate what to draw.
vcount Specifies how many vertices are in vlist.

XDrawFilled draws arbitrary polygons or curves and then fills them.

Associating User Data with a Value

These functions have been superseded by the context management functions (see section 16.10). It is often necessary to associate arbitrary information with resource IDs. Xlib provides the XAssocTable functions that you can use to make such an association. Application programs often need to be able to easily refer to their own data structures when an event arrives. The XAssocTable system provides users of the X library with a method for associating their own data structures with X resources (Pixmaps, Fonts, Windows, and so on).

An XAssocTable can be used to type X resources. For example, the user may want to have three or four types of windows, each with different properties. This can be accomplished by associating each X window ID with a pointer to a window property data structure defined by the user. A generic type has been defined in the X library for resource IDs. It is called an XID.

There are a few guidelines that should be observed when using an XAssocTable:
• All XIDs are relative to the specified display.
• Because of the hashing scheme used by the association mechanism, the following rules for determining the size of a XAssocTable should be followed. Associations will be made and looked up more efficiently if the table size (number of buckets in the hashing system) is a power of two and if there are not more than 8 XIDs per bucket.

To return a pointer to a new XAssocTable, use XCreateAssocTable.

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

To create an entry in a given XAssocTable, use XMakeAssoc.

XMakeAssoc(display, table, x_id, data)
   Display *display;
   XAssocTable *table;
   XID x_id;
   char *data;

display Specifies the connection to the X server.
table Specifies the assoc table.
x_id Specifies the X resource ID.
data Specifies the data to be associated with the X resource ID.

XMakeAssoc inserts data into an XAssocTable keyed on an XID. Data is inserted into the table only once. Redundant inserts are ignored. The queue in each association bucket is sorted from the lowest XID to the highest XID.

To obtain data from a given XAssocTable, use XLookUpAssoc.

char *XLookUpAssoc(display, table, x_id)
   Display *display;
   XAssocTable *table;
   XID x_id;

display Specifies the connection to the X server.
table Specifies the assoc table.
x_id Specifies the X resource ID.

XLookUpAssoc retrieves the data stored in an XAssocTable by its XID. If an appropriately matching XID can be found in the table, XLookUpAssoc returns the data associated with it. If the x_id cannot be found in the table, it returns NULL.

To delete an entry from a given XAssocTable, use XDeleteAssoc.

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

display Specifies the connection to the X server.
table Specifies the assoc table.
x_id Specifies the X resource ID.

XDeleteAssoc deletes an association in an XAssocTable keyed on its XID. Redundant deletes (and deletes of nonexistent XIDs) are ignored. Deleting associations in no way impairs the performance of an XAssocTable.

To free the memory associated with a given XAssocTable, use XDestroyAssocTable.
XDestroyAssocTable(table)
    XAssocTable *table;

    table Specifies the assoc table.
Glossary

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

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

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

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

Background
An InputOutput window can have a background, which is defined as a pixmap. When regions of the window have their contents lost or invalidated, the server automatically tiles those regions with the background.

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

Base font name
A font name used to select a family of fonts whose members may be encoded in various charsets. The CharSetRegistry and CharSetEncoding fields of an XLFD name identify the charset of the font. A base font name may be a full XLFD name, with all fourteen '-' delimiters, or an abbreviated XLFD name containing only the first 13 fields of an XLFD name, up to but not including CharSetRegistry, with or without the thirteenth '-', or a non-XLFD name. Any XLFD fields may contain wild cards.

When creating an XFontSet, Xlib accepts from the client a list of one or more base font names which select one or more font families. They are combined with charset names obtained from the encoding of the locale to load the fonts required to render text.

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

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

Bitmap
A bitmap is a pixmap of depth one.
Border

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

Button grabbing

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

Byte order

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

Character

A member of a set of elements used for the organization, control, or representation of text (ISO2022, as adapted by XPG3). Note that in ISO2022 terms, a character is not bound to a coded value until it is identified as part of a coded character set.

Character glyph

The abstract graphical symbol for a character. Character glyphs may or may not map one-to-one to font glyphs, and may be context-dependent, varying with the adjacent characters. Multiple characters may map to a single character glyph.

Character set

A collection of characters.

Charset

An encoding with a uniform, state-independent mapping from characters to codepoints. A coded character set.

For display in X, there can be a direct mapping from a charset to one font, if the width of all characters in the charset is either one or two bytes. A text string encoded in an encoding such as Shift-JIS cannot be passed directly to the X server, because the text imaging requests accept only single-width charsets (either 8 or 16 bits).Charsets which meet these restrictions can serve as "font charsets." Font charsets strictly speaking map font indices to font glyphs, not characters to character glyphs.

Note that a single font charset is sometimes used as the encoding of a locale, for example, ISO8859-1.

Children

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

Class

Windows can be of different classes or types. See the entries for InputOnly and InputOutput windows for further information about valid window types.

Client

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

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

Coded character

A character bound to a codepoint.

Coded character set

A set of unambiguous rules that establishes a character set and the one-to-one relationship between each character of the set and its bit representation. (ISO2022, as adapted by XPG3) A definition of a one-to-one mapping of a set of characters to a set of codepoints.

Codepoint

The coded representation of a single character in a coded character set.

Colormap

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

Connection

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

Containment

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

Coordinate system

The coordinate system has X horizontal and Y vertical, with the origin [0, 0] at the upper left. Coordinates are integral and coincide with pixel centers. Each window and pixmap has its own coordinate system. For a window, the origin is inside the border at the inside upper-left corner.

Cursor

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

Depth

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

Device

Keyboards, mice, tablets, track-balls, button boxes, and so on are all collectively known as input devices. Pointers can have one or more buttons (the most common number is three). The core protocol only deals with two devices: the keyboard and the pointer.
DirectColor

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

Display

A server, together with its screens and input devices, is called a display. The Xlib Display structure contains all information about the particular display and its screens as well as the state that Xlib needs to communicate with the display over a particular connection.

Drawable

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

Encoding

A set of unambiguous rules that establishes a character set and a relationship between the characters and their representations. The character set does not have to be fixed to a finite pre-defined set of characters. The representations do not have to be of uniform length. Examples are an ISO2022 graphic set, a state-independent or state-dependent combination of graphic sets, possibly including control sets, and the X Compound Text encoding.

In X, encodings are identified by a string which appears as: the CharSetRegistry and CharSetEncoding components of an XLFD name; the name of a charset of the locale for which a font could not be found; or an atom which identifies the encoding of a text property or which names an encoding for a text selection target type. Encoding names should be composed of characters from the X Portable Character Set.

Escapement

The escapement of a string is the distance in pixels in the primary draw direction from the drawing origin to the origin of the next character (that is, the one following the given string) to be drawn.

Event

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

Event mask

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

Event propagation

Device-related events propagate from the source window to ancestor windows until some client has expressed interest in handling that type of event or until the event is discarded explicitly.
Event synchronization
There are certain race conditions possible when demultiplexing device events to clients (in particular, deciding where pointer and keyboard events should be sent when in the middle of window management operations). The event synchronization mechanism allows synchronous processing of device events.

Event source
The deepest viewable window that the pointer is in is called the source of a device-related event.

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

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

Font
A font is an array of glyphs (typically characters). The protocol does no translation or interpretation of character sets. The client simply indicates values used to index the glyph array. A font contains additional metric information to determine interglyph and interline spacing.

Frozen events
Clients can freeze event processing during keyboard and pointer grabs.

Font glyph
The abstract graphical symbol for an index into a font.

GC
GC is an abbreviation for graphics context. See Graphics context.

Glyph
An identified abstract graphical symbol independent of any actual image. (ISO/IEC/DIS 9541-1) An abstract visual representation of a graphic character, not bound to a codepoint.

Glyph image
An image of a glyph, as obtained from a glyph representation displayed on a presentation surface. (ISO/IEC/DIS 9541-1)

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

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

Gravity
The contents of windows and windows themselves have a gravity, which determines how the contents move when a window is resized. See Bit gravity and Window gravity.
GrayScale

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

Host Portable Character Encoding

The encoding of the X Portable Character Set on the host. The encoding itself is not defined by this standard, but the encoding must be the same in all locales supported by Xlib on the host. If a string is said to be in the Host Portable Character Encoding, then it only contains characters from the X Portable Character Set, in the host encoding.

Hotspot

A cursor has an associated hotspot, which defines the point in the cursor corresponding to the coordinates reported for the pointer.

Identifier

An identifier is a unique value associated with a resource that clients use to name that resource. The identifier can be used over any connection to name the resource.

Inferiors

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

Input focus

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

Input manager

Control over keyboard input is typically provided by an input manager client, which usually is part of a window manager.

InputOnly window

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

InputOutput window

An InputOutput window is the normal kind of window that is used for both input and output. InputOutput windows can have both InputOutput and InputOnly windows as inferiors.

Internationalization

The process of making software adaptable to the requirements of different native languages, local customs, and character string encodings. Making a computer program adaptable to different locales without program source modifications or recompilation.

ISO2022

ISO standard for code extension techniques for 7-bit and 8-bit coded character sets.

Key grabbing

Keys on the keyboard can be passively grabbed by a client. When the key is pressed, the keyboard is then actively grabbed by the client.
Keyboard grabbing
A client can actively grab control of the keyboard, and key events will be sent to that client rather than the client the events would normally have been sent to.

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

Latin-1
The coded character set defined by the ISO8859-1 standard.

Latin Portable Character Encoding
The encoding of the X Portable Character Set using the Latin-1 codepoints plus ASCII control characters. If a string is said to be in the Latin Portable Character Encoding, then it only contains characters from the X Portable Character Set, not all of Latin-1.

Locale
The international environment of a computer program defining the “localized” behavior of that program at run-time. This information can be established from one or more sets of localization data. ANSI C defines locale-specific processing by C system library calls. See ANSI C and the X/Open Portability Guide specifications for more details. In this specification, on implementations that conform to the ANSI C library, the “current locale” is the current setting of the LC_CTYPE setlocale category. Associated with each locale is a text encoding. When text is processed in the context of a locale, the text must be in the encoding of the locale. The current locale affects Xlib in its:

- Encoding and processing of input method text
- Encoding of resource files and values
- Encoding and imaging of text strings
- Encoding and decoding for inter-client text communication

Localization
The process of establishing information within a computer system specific to the operation of particular native languages, local customs and coded character sets. (XPG3)

Locale name
The identifier used to select the desired locale for the host C library and X library functions. On ANSI C library compliant systems, the locale argument to the setlocale function.

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

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

Monochrome
Monochrome is a special case of StaticGray in which there are only two colormap entries.

Multibyte
A character whose codepoint is stored in more than one byte; any encoding which can contain multibyte characters; text in a multibyte encoding. The “char *” null-terminated string datatype in ANSI C. Note that references in this document to multibyte strings imply only that the strings may contain multibyte characters.
Obscure
A window is obscured if some other window obscures it. A window can be partially obscured and so still have visible regions. Window A obscures window B if both are viewable InputOutput windows, if A is higher in the global stacking order, and if the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Note the distinction between obscures and occludes. Also note that window borders are included in the calculation.

Occlude
A window is occluded if some other window occludes it. Window A occludes window B if both are mapped, if A is higher in the global stacking order, and if the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Note the distinction between occludes and obscures. Also note that window borders are included in the calculation and that InputOnly windows never obscure other windows but can occlude other windows.

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

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

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

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

Pixmap
A pixmap is a three-dimensional array of bits. A pixmap is normally thought of as a two-dimensional array of pixels, where each pixel can be a value from 0 to $2^N - 1$, and where N is the depth (z axis) of the pixmap. A pixmap can also be thought of as a stack of N bitmaps. A pixmap can only be used on the screen that it was created in.

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

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

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

Pointer grabbing
A client can actively grab control of the pointer. Then button and motion events will be sent to that client rather than the client the events would normally have been sent to.
Pointing device

A pointing device is typically a mouse, tablet, or some other device with effective dimensional motion. The core protocol defines only one visible cursor, which tracks whatever pointing device is attached as the pointer.

POSIX


POSIX Portable Filename Character Set

The set of 65 characters which can be used in naming files on a POSIX-compliant host that are correctly processed in all locales. The set is:

a..z A..Z 0..9 _-

Property

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

Property list

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

PseudoColor

PseudoColor is a class of colormap in which a pixel value indexes the colormap entry to produce an independent RGB value; that is, the colormap is viewed as an array of triples (RGB values). The RGB values can be changed dynamically.

Rectangle

A rectangle specified by \([x,y,w,h]\) has an infinitely thin outline path with corners at \([x,y]\), \([x+w,y]\), \([x+w,y+h]\), and \([x,y+h]\). When a rectangle is filled, the lower-right edges are not drawn. For example, if \(w=h=0\), nothing would be drawn. For \(w=h=1\), a single pixel would be drawn.

Redirecting control

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

Reply

Information requested by a client program using the X protocol is sent back to the client with a reply. Both events and replies are multiplexed on the same connection. Most requests do not generate replies, but some requests generate multiple replies.

Request

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

Resource

Windows, pixmaps, cursors, fonts, graphics contexts, and colormaps are known as resources. They all have unique identifiers associated with them for naming purposes. The lifetime of a resource usually is bounded by the lifetime of the connection over which the resource was created.
RGB values
RGB values are the red, green, and blue intensity values that are used to define a color. These values are always represented as 16-bit, unsigned numbers, with 0 the minimum intensity and 65535 the maximum intensity. The X server scales these values to match the display hardware.

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

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

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

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

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

Screen
A server can provide several independent screens, which typically have physically independent monitors. This would be the expected configuration when there is only a single keyboard and pointer shared among the screens. A Screen structure contains the information about that screen and is linked to the Display structure.

Selection
A selection can be thought of as an indirect property with dynamic type. That is, rather than having the property stored in the X server, it is maintained by some client (the owner). A selection is global and is thought of as belonging to the user and being maintained by clients, rather than being private to a particular window subhierarchy or a particular set of clients. When a client asks for the contents of a selection, it specifies a selection target type, which can be used to control the transmitted representation of the contents. For example, if the selection is "the last thing the user clicked on," and that is currently an image, then the target type might specify whether the contents of the image should be sent in XY format or Z format.

The target type can also be used to control the class of contents transmitted; for example, asking for the "looks" (fonts, line spacing, indentation, and so forth) of a paragraph selection, rather than the text of the paragraph. The target type can also be used for other purposes. The protocol does not constrain the semantics.

Server
The server, which is also referred to as the X server, provides the basic windowing mechanism. It handles IPC connections from clients, multiplexes graphics requests onto the screens, and demultiplexes input back to the appropriate clients.
Server grabbing

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

Shift sequence

ISO2022 defines control characters and escape sequences which temporarily (single shift) or permanently (locking shift) cause a different character set to be in effect ("invoking" a character set).

Sibling

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

Stacking order

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

State-dependent encoding

An encoding in which an invocation of a charset can apply to multiple characters in sequence. A state-dependent encoding begins in an "initial state" and enters other "shift states" when specific "shift sequences" are encountered in the byte sequence. In ISO2022 terms, this means use of locking shifts, not single shifts.

State-independent encoding

Any encoding in which the invocations of the charsets are fixed, or span only a single character. In ISO2022 terms, this means use of at most single shifts, not locking shifts.

StaticColor

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

StaticGray

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

Status

Many Xlib functions return a success status. If the function does not succeed, however, its arguments are not disturbed.

Stipple

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

STRING encoding

Latin-1, plus tab and newline.

String Equivalence

Two ISO Latin-1 STRING8 values are considered equal if they are the same length and if corresponding bytes are either equal or are equivalent as follows: decimal values 65 to 90 inclusive (characters "A" to "Z") are pairwise equivalent to decimal values 97 to 122 inclusive (characters "a" to "z"), decimal values 192 to 214 inclusive (characters "A grave" to "O diaeresis") are pairwise equivalent to decimal values 224 to 246 inclusive (characters "a grave" to "o diaeresis"), and decimal values 216 to 222 inclusive (characters "O oblique" to "THORN") are pairwise equivalent to decimal values 246 to 254 inclusive (characters "o oblique" to "thorn").
Tile

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

Timestamp

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

TrueColor

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

Type

A type is an arbitrary atom used to identify the interpretation of property data. Types are completely uninterpreted by the server. They are solely for the benefit of clients. X predefines type atoms for many frequently used types, and clients also can define new types.

Viewable

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

Visible

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

Whitespace

Any spacing character. On implementations that conform to the ANSI C library, whitespace is any character for which isspace returns true.

Window gravity

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

Window manager

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

A basic set of 97 characters which are assumed to exist in all locales supported by Xlib. This set contains the following characters:

a..z A..Z 0..9
!"#$%&'0*+.-/:;<=>?@[\]^_`{r,
<space>, <tab>, and <newline>

This is the left/lower half (also called the GO set) of the graphic character set of ISO8859-1 plus <space>, <tab>, and <newline>. It is also the set of graphic characters in 7-bit ASCII plus the same three control characters. The actual encoding of these characters on the host is system dependent; see the Host Portable Character Encoding.

XLFD

The X Logical Font Description Conventions that define a standard syntax for structured font names.

XY format

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

Z format

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

References

POSIX: Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language], ISO/IEC 9945-1.
ISO2022: Information processing - ISO 7-bit and 8-bit coded character sets - Code extension techniques.
Index

Atom, 49, 50, 249, 347, 349, 364
getting name, 51
interning, 51
predefined, 49, 260
Authentication, 153
AutoRepeatModeDefault, 213
AutoRepeatModeOff, 213, 214
AutoRepeatModeOn, 213, 214

B
B16, 347
B32, 347
Background, 364
Backing store, 364

BadAccess, 41, 42, 76, 77, 154, 155, 156, 197, 205, 208
BadAtom, 52, 53, 54, 55, 56, 57, 197, 198, 244, 245, 249, 267, 274, 275, 287, 288, 356, 357, 358
BadColor, 31, 42, 43, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 149, 150, 197, 198, 244, 245, 249
BadCursor, 31, 42, 43, 61, 197, 198, 203, 204, 205, 245, 250
BadDrawable, 47, 48, 58, 61, 107, 113, 114, 120, 122, 123, 124, 126, 127, 128, 129, 140, 141, 142, 144, 145, 146, 197, 198, 319, 358
BadFont, 60, 107, 108, 115, 133, 134, 138, 140, 196, 197, 198, 233
BadIDChoice, 197
BadImplementation, 197, 198
BadLength, 197, 198, 220, 347
BadName, 77, 132, 196, 197, 198
BadPixmap, 31, 42, 43, 59, 60, 107, 108, 114, 115, 197, 198, 244, 245, 250
BadRequest, 197, 198
153, 154, 155, 156, 194, 197, 198, 203, 204, 205, 206, 208, 210, 212, 213, 214, 215, 216, 218, 219, 220, 221, 263, 267
Base font name, 364
Below, 35, 36
Bit:
  gravity, 364
  plane, 364
Bitmap, 2, 364
BitmapBitOrder, 15
BitmapFileInvalid, 319
BitmapNoMemory, 319
BitmapOpenFailed, 319
BitmapPad, 15
BitmapSuccess, 319
BitmapUnit, 14
BlackPixel, 8, 9
BlackPixelOfScreen, 16
Bool, 4, 159, 191, 192
Border, 365
Bottomlf, 35, 36, 37
Button1, 165
Button1Mask, 165, 166
Button1Motion, 28
Button1MotionMask, 159, 163, 202
Button2, 165
Button2Mask, 165, 166
Button2Motion, 28
Button2MotionMask, 159, 202
Button3, 165
Button3Mask, 165, 166
Button3Motion, 28
Button3MotionMask, 159, 202
Button4, 165
Button4Mask, 165, 166
Button4Motion, 28
Button4MotionMask, 159, 202
Button5, 165
Button5Mask, 165, 166
Button5Motion, 28
Button5MotionMask, 159, 163, 202
Button:
  grabbing, 204, 365
  ungrabbing, 205
ButtonMotion, 28
ButtonMotionMask, 159, 160, 163, 202
ButtonRelease, 28, 157, 163, 168, 209
ButtonReleaseMask, 159, 163, 202
Byte:
  order, 365
C
CallbackPrototype, 251
CapBut, 102, 104, 105, 111
CapNoLast, 104, 105, 111
CapProjecting, 104, 105, 111
CapRound, 104, 105, 111
CCC, 67
  creation, 82
  default, 67, 79, 80
  freeing, 83
  of colormap, 67, 79, 80
CellsOfScreen, 16
CenterGravity, 27, 46, 47
Changing:
  pointer grab, 204
Character glyph, 365
Character set, 365
Character, 365
Charset, 365
Child window, 1
Child Window, 45
Children, 365
Chroma, 93, 94
  maximum, 93, 94
CIE metric lightness, 90, 91, 92, 93
  maximum, 90, 91, 92
  minimum, 91, 93
CirculateNotify, 40, 157, 160, 165, 174, 179
CirculateRequest, 40, 157, 180, 187
Class, 365
Clearing:
  areas, 118
  windows, 119
Client White Point, 67
  of Color Conversion Context, 81
Client, 365
ClientMessage, 50, 157, 159, 183, 262
ClientWhitePointOfCCC, 81
ClipByChildren, 102, 106, 117, 118
Clipping region, 365
Coded character set, 366
Coded character, 366
Codepoint, 366
Color Characterization Data, 100
Color Conversion Context, 67
  creation, 67, 79, 82
default, 67, 79, 80
freeing, 83
of colormap, 67, 79, 80
color conversion, 83
Color map, 63, 71
Color, 63
allocation, 71
color:
 allocation, 71, 72, 73
conversion, 83
deallocation, 74
naming, 69, 70, 72, 77
querying, 78, 79
storing, 75, 76, 77
Colormap, 2, 20, 21, 249, 347, 366
colormap:
 CCC of, 79, 80
ColormapChangeMask, 159, 182
ColormapInstalled, 183
ColormapNotify, 41, 69, 149, 150, 157, 182
colormaps:
 standard, 286
ColormapUninstalled, 183
Complex, 127, 128
ConfigureNotify, 36, 157, 160, 165, 174, 175, 179
ConfigureRequest, 2, 35, 37, 38, 39, 41, 157, 180, 181, 187
ConfigureRequestEvent, 263
ConfigureWindow, 181, 263
Connection, 366
ConnectionNumber, 9
Containment, 366
Control, 220
ControlMask, 165, 166, 202, 205
ConvertSelection, 185
Convex, 127, 128
Coordinate system, 366
CoordModeOrigin, 122, 123, 127
CoordModePrevious, 122, 123, 127, 128
CopyArea, 120
CopyFromParent, 25, 26, 29, 30, 41, 43
Copying:
 areas, 119
planes, 120
CreateNotify, 31, 32, 157, 160, 174, 175, 176
CurrentTime, 56, 57, 160, 185, 186, 195, 201, 202, 203, 204, 206, 207, 208, 211, 375
Cursor, 2, 20, 21, 250, 347, 366
 Initial State, 31
limitations, 61
CursorShape, 112, 113
Cut Buffers, 312
CWB ackingPixel, 24
CWB ackingPlanes, 24
CWB ackingStore, 24
CWB ackPixel, 24
CWB ackPixmap, 24
CWB bitGravity, 24
CWB orderPixel, 24
CWB orderPixmap, 24
CWB orderWidth, 35
CWB oloormap, 24
CWC ursor, 24
CWD ontPropagate, 24
CWE ventMask, 24
CWH eight, 35
CW OverrideRedirect, 24
CWS aveUnder, 24
CWSibling, 35
CWStackMode, 35
CWW idth, 35
CWW inGravity, 24
CWX, 35
CWY, 35

D
Data16, 350
Data32, 350
Data, 350
Debugging:
 error event, 196
 error handlers, 196
 error message strings, 198
 error numbers, 197
 synchronous mode, 196
Default Protection, 154
Default Blanking, 152, 153
Default Colormap, 9, 63
DefaultColormapOfScreen, 16
Default Depth, 10
DefaultDepthOfScreen, 17
Default Exposures, 152, 153
DefaultGC, 10
DefaultGOFScreen, 17
Default RootWindow, 10
Default Screen, 7, 8, 11
Default Screen Of Display, 10
Default Visual, 11, 63
Default Visual Of Screen, 17
Depth, 366
Destination, 103
DestroyAll, 20, 21
DestroyNotify, 32, 157, 160, 174, 176
Device Color Characterization, 99
device profile, 67, 100
Device, 366
DirectColor, 22, 23, 63, 68, 73, 74, 286, 366, 375

379
EnableAccess, 156
Display Functions, 103
Display, 7, 8, 20, 220, 344, 345, 349, 350, 367, 373
data structure, 8
structure, 367, 373
DisplayCells, 11
DisplayHeight, 15
DisplayHeightMM, 15
DisplayOfCCC, 80
DisplayOfScreen, 17
DisplayPlanes, 11
DisplayString, 12
DisplayWidth, 15
DisplayWidthMM, 16
DoBlue, 63, 70, 75, 76, 77, 78
DoesBackingStore, 17
DoesSaveUnders, 17
DoGreen, 63, 70, 75, 76, 77, 78
DontAllowExposures, 152, 153
DontPreferB tanking, 152, 153
DoRed, 63, 70, 75, 76, 77, 78
Drawable, 2, 347, 349, 367
Drawing:
arcs, 124
image text, 141
lines, 122
points, 121
polygons, 122
rectangles, 123
strings, 140
text items, 139

E
EastGravity, 27, 46, 47
EnableAccess, 156
Encoding, 367
EnterNotify, 157, 165, 166, 167, 168, 169, 171, 179, 203
EnterWindowMask, 159, 165, 202
Environment:
DISPLAY, 7
Error:
codes, 197
handlers, 196
handling, 3
Escapement, 367
EvenOddRule, 102, 106, 107, 112, 309
event mask, 159
Event, 2, 157, 367
categories, 157
Exposure, 368
mask, 367
propagation, 187, 367
source, 368
synchronization, 367
types, 157
EventMaskOfScreen, 18
Events:
ButtonPress, 163
ButtonRelease, 163
CirculateNotify, 174
CirculateRequest, 180
ClientMessage, 183
ColormapNotify, 182
ConfigureNotify, 175
ConfigureRequest, 181
CreateNotify, 175
DestroyNotify, 176
EnterNotify, 165
Expose, 172
FocusIn, 168
FocusOut, 168
GraphicsExpose, 173
GravityNotify, 177
KeymapNotify, 171
KeyPress, 163
KeyRelease, 163
LeaveNotify, 165
MapNotify, 177
MappingNotify, 178
MapRequest, 181
MotionNotify, 163
NoExpose, 173
PropertyNotify, 184
ReparentNotify, 178
ResizeRequest, 182
SelectionClear, 184
SelectionNotify, 185
SelectionRequest, 185
UnmapNotify, 179
VisibilityNotify, 179
Expose, 2, 24, 26, 27, 28, 32, 33, 34, 35, 37, 38, 39, 118, 147, 152, 157, 165, 169, 172, 173, 179
ExposureMask, 159, 172
Extension, 368

F
FamilyChaos, 154
FamilyDECnet, 154
FamilyInternet, 154
Files:
  $HOME/.Xdefaults, 359
  /etc/tyts, 212
  /etc/X7. hosts, 154
  <sys/socket.h>, 155
  <X11/Xlib.h>, 337, 344
  <X11/Xlibint.h>, 336, 346, 347, 349
  <X11/Xproto.h>, 346, 347, 348, 349
  <Xproto.h>, 348

Filling:
  arcs, 128
  polygon, 127
  rectangles, 126
FillOpaqueSupplied, 105, 106, 112, 120
FillPolygon, 105, 106, 112
FillStippled, 105, 106, 112
FillTiled, 106, 112
FlushGC, 345, 346
FocusChangeMask, 159, 168
FocusIn, 157, 168, 169, 170, 171, 206, 207, 211, 212
FocusOut, 157, 165, 168, 169, 170, 171, 179, 206, 207, 211, 212
Font glyph, 368
Font, 2
  font, 6
Font, 20, 21, 129, 347, 368
FontLeftToRight, 130, 136, 137, 138
FontRightToLeft, 130, 136, 137, 138
Fonts, 361
  freeing font information, 132
  getting information, 132
  unloading, 132
Forget, 27
ForgetGravity, 25, 27, 46
fork, 12
Freeing:
  colors, 74
  resources, 24, 42, 43
Frozen events, 368
function set, 99
  LINEAR_RGB, 99

G

gamut compression, 67
  client data, 81
  procedure, 81
  setting in Color Conversion Context, 81
gamut handling, 67
gamut querying, 88
GC, 344, 368
GCArcMode, 101, 108
GCBackground, 101, 108
GC CapStyle, 101, 108
GC ClipMask, 101, 108
GC ClipXOrigin, 101, 108
GC ClipYOrigin, 101
GC ClipYOrigin, 108
GCDashList, 101, 108
GCDashOffset, 101, 108
GC FillRule, 101, 108
GC FillStyle, 101, 108
GCForeground, 101, 108
GCFont, 101, 108
GC Function, 101, 108
GGraphicsExposures, 101, 108
GGlyph, 368
GCLineStyle, 101, 108
GCLineWidth, 101, 108
GContext, 2, 20, 21, 109, 132, 137, 138, 197, 347
GC PlaneMask, 101, 108
GCSupple, 101, 108
GCSubwindowMode, 101, 108
GC Tile, 101, 108
GCTileSüpXOrigin, 101, 108
GCTileStipYOrigin, 101, 108
GeometryCallback, 251
GetEmptyReq, 349
GetKeyboardControl, 348, 351
GetReq, 349
GetReqExtra, 349
GetResReq, 349
GetWindowAttributes, 348, 350, 351
Glyph image, 368
Glyph, 368
Grab, 368
Grabbing:
  buttons, 204
  keyboard, 206
  keys, 207
  pointer, 202
  server, 151
GrabFrozen, 203, 206
GrabInvalidTime, 203, 206
GrabModeAsync, 162, 202, 203, 204, 206, 207
GrabModeSync, 202, 203, 204, 206, 207
GrabNotViewable, 203, 206
GrabSuccess, 202
Graphics context, 101, 368
  initializing, 107
  path, 104
GraphicsExpos, 107, 117, 119, 157, 158, 159, 173, 174
Gravity, 368
Gravity Notify, 27, 36, 157, 160, 165, 174, 177, 179
Gray Scale, 22, 23, 60, 63, 68, 73, 285, 286, 357.
Xlib - C Library

369, 374
GXand, 103
GXandInverted, 103
GXandReverse, 103
GXclear, 103
GXcopy, 102, 103, 119, 142
GXcopyInverted, 103
GXequiv, 103
GXinvert, 103
GXnand, 103
GXnoop, 103
GXnor, 103
GXor, 103
GXorInverted, 103
GXorReverse, 103
GXset, 103
GXxor, 103

H

Hash Lookup, 361
HeightMMOfScreen, 18
HeightOfScreen, 18
HeightValue, 308
Host Portable Character Encoding, 369
Hotspot, 369

I

IconicState, 262, 271
IconMaskHint, 270
IconPixmapHint, 270
IconPositionHint, 270
IconWindowHint, 270
Identifier, 369
Image text:
  drawing, 141
ImageByteOrder, 14
IncludeInferiors, 106, 117
Inferiors, 369
InitExtension, 199
Input Control, 157
Input:
  focus, 369
  manager, 369
InputFocus, 194
InputHint, 270
InputOnly, 24, 25, 27, 28, 29, 30, 31, 32, 35, 42,
  43, 46, 48, 58, 113, 114, 118, 119, 148, 172, 179,
  197, 365, 367, 369, 371
InputOutput, 23, 24, 25, 26, 27, 28, 29, 30, 31,
  32, 33, 46, 106, 364, 365, 369, 370
Internationalization, 369
IsCursorKey, 306
IsFunctionKey, 306

J

JoinBevel, 105, 111
JoinMiter, 102, 105, 111
JoinRound, 105, 111

K

KBAutoRepeatMode, 212
KBBellDuration, 212
KBBellPercent, 212
KBBellPitch, 212
KBKey, 212
KBKeyIdClickPercent, 212
KBLed, 212
KBLedMode, 212
Key:
  grabbing, 207, 369
  ungrabbing, 208
Keyboard:
  bell volume, 212
  bit vector, 212
  grabbing, 206, 370
  keyclick volume, 212
  ungrabbing, 206
KeymapNotify, 157, 171
KeymapStateMask, 159, 171
KeyMapStateMask, 202
KeyPress, 28, 157, 163, 171, 206, 207, 209, 213,
  217, 257, 258, 304
KeyPressMask, 159, 163
KeyRelease, 28, 157, 163, 171, 206, 209, 213,
  257, 304
KeyReleaseMask, 159, 163
Keysym, 370

L

LastKnownRequestProcessed, 12
Latin Portable Character Encoding, 370
Latin-1, 370
LeaveNotify, 157, 165, 166, 167, 168, 169, 179,
  203
LeaveWindowMask, 159, 165, 202
LedModeOff, 213

382
LedModeOn, 213
LineDoubleDash, 104, 106, 111
LineOnOffDash, 104, 106, 111
Lines:
  drawing, 122
LineSolid, 102, 104, 105, 111
Locale name, 370
Locale, 370
Localization, 370
Lock, 220
LockDisplay, 348
LockMask, 165, 166, 202, 205
LookupColor, 196
LowerHighest, 39, 40
LSBFirst, 14, 15

M

MapNotify, 33, 157, 160, 165, 174, 177, 179
Mapped window, 370
MappingBusy, 215, 221
MappingFailed, 217, 221
MappingKeyboard, 178, 305
MappingModifier, 178, 305
MappingNotify, 157, 159, 174, 178, 215, 219, 220, 305
MappingPointer, 178
MappingSuccess, 215, 220
MapRequest, 33, 157, 180, 181, 182, 187
MapWindow, 21, 147
MaxCmapsOfScreen, 19
Mblen, 225
Mbtowc, 225
Menus, 151
MinCmapsOfScreen, 19
Mod1, 220
Mod1Mask, 165, 166, 202, 205
Mod2, 220
Mod2Mask, 165, 166, 202, 205
Mod3, 220
Mod3Mask, 165, 166, 202, 205
Mod4, 220
Mod4Mask, 165, 166, 202, 205
Mod5, 220
Mod5Mask, 165, 166, 202, 205
Modifier keys, 370
Monochrome, 370
MotionNotify, 157, 160, 163, 165, 195
Mouse:
  programming, 212
MSBFirst, 14, 15
Multibyte, 370

N

NextRequest, 12, 197
NoEventMask, 28, 29, 159
NoExpose, 117, 120, 157, 159, 173
Nonconvex, 127, 128
None, 4, 25, 26, 29, 41, 42, 43, 47, 48, 49, 51,
53, 56, 57, 59, 60, 69, 102, 106, 115, 116, 118,
119, 139, 140, 162, 165, 166, 170, 171, 175, 181,
183, 185, 186, 202, 203, 204, 210, 211, 212, 234,
256, 288, 320, 354
NoOperation, 19
NormalState, 271
NorthEastGravity, 27, 46, 47, 309
NorthGravity, 27, 46, 47
NorthWestGravity, 25, 27, 46, 47, 309
NoSymbol, 218, 219, 304, 305
NotifyAncestor, 166, 167, 169
NotifyDetailNone, 169, 170, 171
NotifyGrab, 166, 167, 168, 171
NotifyHint, 163, 165
NotifyInferior, 166, 167, 169
NotifyNormal, 165, 166, 168, 169
NotifyPointer, 169, 170, 171
NotifyPointerRoot, 169, 170, 171
NotifyUngrab, 166, 167, 168, 171
NotifyVirtual, 166, 167, 169
NotifyWhileGrabbed, 168, 169
NotUseful, 17, 25, 28, 47
NULLQUARK, 301

O

Obscure, 370
Occlude, 371
OpenFont, 196
Opposite, 35, 36, 37
Output Control, 157
OwnerGrabButtonMask, 159, 162

P

PackData, 353
Padding, 371
PAAllHints, 272, 273
Parent Window, 1, 45
ParentRelative, 24, 25, 26, 41, 42, 148
PAspect, 272
Passive grab, 201, 371
PBaseSize, 272
Pixel value, 103, 371
Pixmap, 2, 20, 21, 250, 347, 349, 371
Pixmaps, 361
PlaceOnBottom, 174, 181
PlaceOnTop, 174, 181
Plane, 371
  copying, 120
  mask, 103, 371
PlanesOfScreen, 19
PMaxSize, 272
PMinSize, 272
Pointer, 371
  grabbing, 202, 204, 371
  ungrabbing, 203
PointerMotion, 28
PointerMotionHintMask, 159, 163, 202
PointerMotionMask, 159, 160, 163, 202
PointerRoot, 21, 170, 171, 211, 212, 271
PointerWindow, 194
Pointing device, 371
Points:
  drawing, 121
Polygons:
  drawing, 122
  filling, 127
PolyLine, 123, 124
POSIX Portable Filename Character Set, 372
POSIX System Call:
  fork, 12
POSIX, 372
PPosition, 272
PreeditCaretCallback, 254
PreeditDoneCallback, 252
PreeditDrawCallback, 252
PreeditStartCallback, 252
PreferBlanking, 152, 153
PResizelnc, 272
Property list, 372
Property, 372
  appending, 54
  changing, 54
  deleting, 55
  format, 54
  getting, 52
  listing, 53
  prepending, 54
  replacing, 54
  type, 54
PropertyChangeMask, 159, 184
PropertyDelete, 184
PropertyNewValue, 184
PropertyNotify, 53, 54, 55, 157, 183, 184
PropModeAppend, 54
PropModePrepend, 54
PropModeReplace, 54

Protocol:
  DECnet, 8
  TCP, 8
ProtocolRevision, 13
ProtocolVersion, 12
PseudoColor, 22, 23, 63, 68, 73, 74, 286, 369, 372, 374
PSize, 272
Psychometric Chroma, 90, 91, 92
  maximum, 90, 91, 92
Psychometric Hue Angle, 90, 91, 92, 93
PWD, 272
Q
QLength, 13
QueryFont, 196, 348, 350
QueryKeymap, 348, 350
QueuedAfterFlush, 188, 189
QueuedAfterReading, 188, 189
QueuedAlready, 188, 189
R
RaiseLowest, 39, 40
read-only colormap cells, 70, 71
  allocating, 71, 72
read/write colormap cells, 70
  allocating, 73
read/write colormap planes:
  allocating, 73
Rectangle, 372
  filling, 126
RectangleIn, 312
RectangleOut, 312
RectanglePart, 312
Rectangles:
  drawing, 123
Redirecting control, 372
Region, 309
ReleaseByFreeingColormap, 285
ReparentNotify, 147, 157, 160, 174, 178
ReplayKeyboard, 208, 209, 210
ReplayPointer, 208, 209, 210
Reply, 372
Request, 372
Requests, 157
Resource IDs, 2, 20, 361
  Cursor, 2
  Font, 2
  freeing, 24, 42, 43
GContext, 2
Pixmap, 2
Window, 2
Resource, 372
ResourceName, 300
RetainPermanent, 20, 21, 152
RetainTemporary, 20, 21, 152
RevertToNone, 211, 212
RevertToParent, 211, 212
RevertToPointerRoot, 211, 212
RGB values, 372
Root, 101, 373
RootWindow, 13, 237
RootWindowOfScreen, 19

S

Save set, 373
Save Unders, 28
Scanline, 373
order, 373
Screen White Point, 88
Screen, 1, 7, 8, 16, 17, 18, 19, 70, 84, 344, 373
structure, 373
ScreenCount, 13
ScreenFormat, 344
ScreenNumberOfCCC, 81
ScreenOfDisplay, 11
ScreenSaverActive, 153
ScreenSaverReset, 152, 153
ScreenWhitePointOfCCC, 81
Selection, 55, 373
converting, 57
getting the owner, 56
setting the owner, 56
SelectionClear, 56, 157, 159, 183, 184
SelectionNotify, 57, 157, 159, 183, 185, 194
SelectionRequest, 56, 57, 157, 159, 183, 185
SendEvent, 158, 340
Serial Number, 197
Server, 373
grabbing, 151, 373
ServerVendor, 13
setlocale, 222, 224, 370
SetModeDelete, 148
SetModeInsert, 148
Shift sequence, 374
Shift, 220
ShiftMask, 165, 166, 202, 205
Sibling, 374
Source, 103
SouthEastGravity, 27, 46, 47, 309
SouthGravity, 27, 46, 47
SouthWestGravity, 27, 46, 47, 309
special, 6
Stacking order, 1, 374

Standard Colormaps, 286
State-dependent encoding, 374
State-independent encoding, 374
StateHint, 270
StaticColor, 22, 23, 63, 68, 374
StaticGravity, 27, 46, 47
StaticGray, 22, 23, 60, 63, 68, 370, 374
Status, 2, 67, 374
StatusDoneCallback, 255
StatusDrawCallback, 256
StatusStart, 256
StatusStartCallback, 255
stdio.h, 353
Stipple, 374
StippleShape, 112, 113
STRING encoding, 374
String Equivalence, 374
Strings:
drawing, 140
strlen, 300
strtok, 225
StructureNotify, 174, 175, 176, 177, 178, 179
StructureNotifyMask, 159, 174, 175, 176, 177, 178, 179
SubstructureNotify, 174, 175, 176, 177, 178, 179
SubstructureNotifyMask, 159, 160, 174, 175, 176, 177, 178, 179, 262
SubstructureRedirectMask, 29, 33, 35, 36, 37, 38, 39, 40, 41, 159, 180, 181, 182, 187, 262
Success, 53, 263, 264, 265
SyncBoth, 208, 209
SyncHandle, 352
SyncKeyboard, 208, 209, 210
SyncPointer, 208, 209, 210

T

Text:
drawing, 139
this, 6
Tile, 2, 375
mode, 24
pixmaps, 24
TileShape, 112, 113
time, 201
Timestamp, 375
TopIf, 35, 36, 37
True, 4, 17, 28, 29, 33, 47, 48, 49, 53, 73, 74, 77, 83, 84, 86, 98, 102, 118, 119, 120, 131, 133, 147, 158, 159, 162, 163, 166, 173, 175, 176, 177, 179, 183, 184, 188, 190, 191, 192, 193, 194, 196, 202, 206, 216, 222, 229, 256, 257, 265, 271, 296, 297, 298, 301, 306, 312, 336, 341, 342, 351
TrueColor, 22, 23, 63, 68, 375

385
Type, 375

U

Ungrabbing:
  buttons, 205
  keyboard, 206
  keys, 208
  pointer, 203
UngrabKeyboard, 207
UngrabPointer, 203
UnlockDisplay, 349
UnmapGravity, 27, 47, 179
UnmapNotify Event, 34, 35
UnmapNotify, 27, 34, 35, 157, 160, 165, 169, 174, 179, 262
UnmapWindow, 147
Unsorted, 116
USPosition, 272
USSize, 272

V

Value, 93, 94, 95
  maximum, 93, 94
  minimum, 95
VendorRelease, 14
Vertex, 359, 360
VertexCurved, 360
VertexDontDraw, 360
VertexDrawLastPoint, 359
VertexEndClosed, 359, 360
VertexRelative, 360
VertexStartClosed, 359, 360
Viewable, 375
VisibilityChangeMask, 159, 179
VisibilityFullyObscured, 180
VisibilityNotify, 30, 157, 165, 169, 174, 179, 180
VisibilityPartiallyObscured, 180
VisibilityUnobscured, 180
Visible, 375
Visual Classes:
  GrayScale, 22
  PseudoColor, 22
  StaticColor, 22
  StaticGray, 22
  TrueColor, 22
Visual Type, 22
Visual, 22, 23, 46, 316, 344
VisualAllMask, 314
VisualBitsPerRGBMask, 314
VisualBlueMaskMask, 314
VisualClassMask, 314
VisualColormapSizeMask, 314
VisualDepthMask, 314
VisualGreenMaskMask, 314
VisualID, 347
VisualIDMask, 314
VisualNoMask, 314
VisualOfCCC, 80
VisualRedMaskMask, 314
VisualScreenMask, 314

W

wctomb, 225
WestGravity, 27, 46, 47
WhenMapped, 17, 28, 41, 47, 172
white point adjustment, 67
  client data, 82
  procedure, 82
  setting in Color Conversion Context, 82
white point, 67
WhitePixel, 8, 9
WhitePixelOfScreen, 16
Whitespace, 375
WidthMMOfScreen, 18
WidthOfScreen, 18
WidthValue, 308
WindingRule, 106, 107, 112, 309
Window, 2, 20, 23, 248, 347, 349
  attributes, 23
  background, 42
  clearing, 119
  defining the cursor, 43
  determining location, 308, 358
  gravity, 375
  icon name, 269
  IDs, 361
InputOnly, 30, 369
InputOutput, 369
manager, 375
managers, 151
mapping, 24
name, 268
parent, 371
root, 373
RootWindow, 13
undefining the cursor, 44
XRootWindow, 13
WindowGroupHint, 270
Windows, 361
WithdrawnState, 271

X

X Portable Character Set, 375
Xlib – C Library

X10 compatibility:
XDraw, 359
XDrawDashed, 359
XDrawFilled, 359, 361
XDrawPatterned, 359
XDrawTiled, 359, 361
X11/cursorfont.h, 3
X11/keysym.h, 4, 217
X11/keysymdef.h, 3, 4, 217, 305
X11/X.h, 2, 3, 103, 157, 159
X11/X10.h, 4, 360
X11/XAtom.h, 3, 50, 133, 260, 286
X11/Xcms.h, 3, 62
X11/Xlib.h, 3, 4, 8, 62, 71, 73, 75, 87
X11/Xlibint.h, 3, 4, 8, 62, 71, 73, 75, 87
X11/Xproto.h, 3, 4, 173, 197, 346, 347, 348, 349
X11/Xprotostr.h, 3, 4
X11/Xresource.h, 3, 290
X11/Xutil.h, 3, 270, 272, 275, 308, 309, 314, 316, 321, 355
XActivateScreenSaver, 153
XAddExtension, 338
XAddHost, 154
XAddHosts, 154, 155
XAddPixel, 317, 318
XAddToExtensionList, 344
XAddToSaveSet, 148
XAllocClassHint, 275, 276, 280, 282
XAllocateColorCells, 68, 71, 73, 75, 78, 79
XAllocateColorPlanes, 68, 71, 73, 75, 78, 79
XAllocateNamedColor, 65, 68, 72, 73, 75
XAllocateSizeHints, 272
XAllocateStandardColormap, 284
XAllocateWMHints, 270
XAllowEvents, 201, 203, 206, 208, 209, 210
XAllocatePlanes, 8
XAnyEvent, 158
XAppendVertex, 359
XArc, 121
XAssocTable, 361, 362
XAutoRepeatOff, 214
XAutoRepeatOn, 214
XBaseFontNameListOfFontSet, 226, 228
XBell, 214
XBitmapBitOrder, 15
XBitmapPad, 15
XBitmapUnit, 14
XBlackPixel, 9
XBlackPixmapOfScreen, 16
XBufferOverflow, 258
XButtonEvent, 163
XButtonPressedEvent, 163, 165
XButtonReleasedEvent, 163, 165
XCellsOfScreen, 16
XChangeActivePointerGrab, 162, 204
XChangeKeyboardControl, 212, 213, 214
XChangeKeyboardMapping, 178, 218, 219
XChangePointerControl, 215, 216
XChangeProperty, 54, 184
XChangeSaveSet, 148
XChangeWindowAttributes, 24, 41, 42, 69, 149, 182, 187
XChar2b, 130, 137, 140
XCharStruct, 129, 130, 131, 132, 136, 137, 138
XCheckIfEvent, 190, 191
XCheckMaskEvent, 192
XCheckTypedEvent, 192, 193
XCheckTypedWindowEvent, 193
XCheckWindowEvent, 191, 192
X CirculateEvent, 174
X CirculateRequestEvent, 180
X CirculateSubwindows, 39, 40, 174, 180
X CirculateSubwindowsDown, 40, 174, 180
X CirculateSubwindowsUp, 40, 174, 180
X ClassHint, 275, 276, 280, 281, 282
X ClearArea, 118, 119
X ClearVertexFlag, 359
X ClearWindow, 41, 119
X ClientMessageEvent, 183
X ClipBox, 310
X CloseDisplay, 19, 20, 338
X CloseIM, 238, 242
XcmsAddColorSpace, 85, 87, 95, 96
XcmsAddFunctionSet, 95, 99
XcmsAllocColor, 71, 87
XcmsAllocNamedColor, 65, 72, 73, 75
XcmsCIELab, 64, 87
XcmsCIELabClipab, 85
XcmsCIELabClipL, 85
XcmsCIELabClipLuv, 85
XcmsCIELabFormat, 63, 98
XcmsCIELabQueryMaxC, 85, 91, 92
XcmsCIELabToCIEXYZ, 98
XcmsCIELabWhiteShiftColors, 87
XcmsCIELuv, 65, 87
XcmsCIELuvClipL, 85
XcmsCIELuvClipLuv, 85
XcmsCIELuvClipuv, 85
XcmsCIELuvFormat, 63, 98
XcmsCIELuvQueryMaxC, 85, 91, 92

387
XcmsCIELuvQueryMaxL, 92
XcmsCIELuvQueryMaxLC, 92, 93
XcmsCIELuvQueryMinL, 93
XcmsCIELuvToCIEuvY, 98
XcmsCIELuvWhiteShiftColors, 87
XcmsCIELuvY, 64, 87
XcmsCIELuvYFormat, 63, 81, 98
XcmsCIELuvYToCIELuv, 98
XcmsCIELuvYToCIEXYZ, 98
XcmsCIELuvYToTekHVC, 98
XcmsCIExyY, 64
XcmsCIExyYFormat, 63, 81, 98
XcmsCIExyYToCIEXYZ, 98
XcmsCIElabFormat, 63, 81, 98
XcmsCIElabToCIELuv, 98
XcmsCIElabY, 64
XcmsCIElabYFormat, 63, 81, 98
XcmsCIExyY, 64
XcmsCIExyYFormat, 63, 81, 98
XcmsCIExyYToCIEXYZ, 98
XcmsCIEXYZFormat, 63, 81, 98
XcmsCIEXYZToCIELab, 98
XcmsCIEXYZToCIELuv, 98
XcmsCIEXYZToCIExyY, 98
XcmsCIEXYZToRGBi, 98
XcmsClientWhitePointOfCCC, 81
XcmsColor, 63, 73, 76, 77, 78, 79, 83, 84, 86, 95, 97, 98
XcmsColorFormat, 63
XcmsColorSpace, 96, 97, 98, 99, 100
XcmsColorSpaces, 96
XcmsCompressionProc, 84
XcmsConvertColors, 83
XcmsCreateCCC, 82, 83
XcmsDefaultCCC, 80
XcmsDisplayOfCCC, 80
XcmsFailure, 67, 70, 71, 82, 83, 91, 92, 93, 100
XcmsFormatOfPrefix, 64, 96, 99
XcmsFreeCCC, 80, 82, 83
XcmsFunctionSet, 99, 100
XcmsInitFailure, 100
XcmsInitNone, 100
XcmsInitSuccess, 100
XcmsLookupColor, 65, 70, 73
XcmsPad, 65
XcmsParseStringProc, 97
XcmsPerScmlnfo, 99, 100
XcmsPrefixOfFormat, 64, 96, 99
XcmsQueryBlack, 88
XcmsQueryBlue, 88, 89
XcmsQueryColor, 76, 78, 79
XcmsQueryColors, 77, 79
XcmsQueryGreen, 89
XcmsQueryRed, 89
XcmsQueryWhite, 89, 90
XcmsRGB, 64, 65, 87
XcmsRGBFormat, 63, 71, 73, 84, 98
XcmsRGBi, 64, 65, 84, 87
XcmsRGBiFormat, 63, 84, 98
XcmsRGBiToCIEXYZ, 98
XcmsRGBiToRGB, 98
XcmsRGBToRGBi, 98
XcmsScreenFreeProc, 100
XcmsScreenInitProc, 100
XcmsScreenNumberOfCCC, 81
XcmsScreenWhitePointOfCCC, 81
XcmsSetCCCOfColormap, 79, 80
XcmsSetCompressionProc, 81, 82
XcmsSetWhiteAdjustProc, 82
XcmsSetWhitePoint, 81
XcmsStoreColor, 76
XcmsStoreColors, 76, 77
XcmsSuccess, 67, 70, 76, 77, 79, 96, 100
XcmsSuccessWithCompression, 67, 70
XcmsTekHVC, 65, 87
XcmsTekHVCClipC, 86
XcmsTekHVCClipV, 86
XcmsTekHVCClipVC, 86
XcmsTekHVCFormat, 63, 98
XcmsTekHVCQueryMaxC, 88, 93
XcmsTekHVCQueryMaxVC, 94
XcmsTekHVCQueryMaxV, 94
XcmsTekHVCQueryMaxYSamples, 94, 95
XcmsTekHVCQueryMinV, 95
XcmsTekHVCQueryMinY, 95
XcmsTekHVCWhiteShiftColors, 87
XcmsUndefinedFormat, 63, 70, 72, 81, 96
XcmsVisualOfCCC, 80
XcmsWhiteAdjustProc, 86
XcNOENT, 321
XCNOMEM, 321
XColor, 61, 63, 65, 71, 75, 76, 77, 78
XColorMapEvent, 183
XCompositeStatus, 307
XCompoundTextStyle, 264
XConfigureEvent, 175
XConfigureRequestEvent, 181
XConfigureWindow, 35, 36, 37, 175, 177, 181, 182
XConnectionNumber, 9
XContextDependentDrawing, 229
XConverterNotFoundError, 263, 265
XConvertSelection, 57, 185
XCopyArea, 107, 117, 119, 120, 159, 173
XCopyColormapAndFree, 68, 69
XCopyGC, 107, 108
XCopyPlane, 58, 107, 117, 119, 120, 159, 173
XCreateAssocTable, 361
XCreateBitmapFromData, 320
XCreateColormap, 67, 68, 75, 285
XCreateFontCursor, 59, 335
XCreateFontSet, 224, 225, 226, 227, 228, 231, 232, 234, 235, 265, 266
XCreateGC, 101, 107, 108, 116, 173, 309, 349
XCreateGlyphCursor, 59, 60
XCreateIC, 224, 238, 244, 246

388
Xlib - C Library
Xll, Release 5

XCreateImage, 316, 318
XCreatePixmap, 58, 349
XCreatePixmapCursor, 60
XCreatePixmapFromBitmapData, 319, 320
XCreateRegion, 309
XCreateSimpleWindow, 30, 31, 32, 149, 176
XCreateWindow, 24, 30, 31, 41, 69, 149, 176, 187
XCreateWindowEvent, 176
XCrossingEvent, 165
XDefaultColormap, 9
XDefaultColormapOfScreen, 16
XDefaultDepth, 10
XDefaultDepthOfScreen, 17
XDefaultGC, 10
XDefaultGCOfScreen, 17
XDefaultRootWindow, 10
XDefaultScreen, 7, 8, 11
XDefaultScreenOfDisplay, 10
XDefaultString, 265
XDefaultVisual, 11
XDefaultVisualOfScreen, 17
XDefineCursor, 31, 43, 44
XDeleteAssoc, 362
XDeleteContext, 321
XDeleteModifiermapEntry, 220
XDeleteProperty, 55, 184
XDestroyAssocTable, 362
XDestroyIC, 244
XDestroyImage, 316, 318
XDestroyRegion, 310
XDestroySubwindows, 32, 176
XDestroyWindow, 32, 176
XDestroyWindowEvent, 176
XDisableAccessControl, 156
XDisplayCells, 8, 11
XDisplayHeight, 8, 15
XDisplayHeightMM, 8, 15
XDisplayKeycodes, 217, 218, 219
XDisplayMotionBufferSize, 195
XDisplayName, 199
XDisplayOfIM, 243
XDisplayOfScreen, 17
XDisplayPlanes, 8, 11
XDisplayString, 12
XDisplayWidth, 8, 15
XDisplayWidthMM, 8, 16
XDoesBackingStore, 17
XDoesSaveUnders, 17
xDosomethingReply, 348
xDosomethingReq, 347, 349
XDraw, 359, 360
XDrawArc, 106, 124, 125, 126
XDrawArcs, 124, 125, 126
XDrawDashed, 359
XDrawFilled, 359, 361
XDrawImageString16, 141, 142
XDrawImageString, 139, 141, 142, 233
XDrawLine, 105, 118, 122, 123
XDrawLines, 122, 123, 359
XDrawPatterned, 359
XDrawPoint, 118, 121, 122, 345
XDrawPoints, 12, 121, 122
XDrawRectangle, 106, 118, 123, 124
XDrawRectangles, 12, 123, 124
XDrawSegments, 12, 105, 122, 123, 359
XDrawString16, 140, 141
XDrawString, 140, 141, 233
XDrawText16, 105, 138, 139, 140
XDrawText, 105, 138, 139, 140, 233
XDrawTiled, 359, 361
XDataObject, 344
XEHeadOfExtensionList, 344
XEmptyRegion, 311, 312
XEnableAccessControl, 156
XEnterWindowEvent, 165, 166, 167, 168
XEqualRegion, 312
XErrorEvent, 196, 197, 342, 343
XESetCloseDisplay, 338
XESetCopyGC, 338, 343
XESetCreateFont, 339
XESetCreateGC, 338
XESetError, 342
XESetErrorString, 342, 343
XESetEventToWire, 341
XESetFlushGC, 343
XESetFreeFont, 339
XESetFreeGC, 339
XESetPrintErrorValues, 343
XESetWireToError, 341, 343
XESetWireToEvent, 340
XEvent, 158, 159, 189, 190, 191, 192, 193, 194, 256, 340
xEvent, 340
xEvent, 341
xEvent, 341
XEvent, 342, 343
XEventMaskOfScreen, 18
XEventsQueued, 13, 188, 189
XExposeEvent, 172
XExtCodes, 337, 338, 353
XExtData, 344
XExtentsOfFontSet, 224, 230
XFetchBuffer, 313
XFetchBytes, 313
XFetchName, 268, 269
XFillArc, 105, 118, 128, 129
XFillAres, 12, 107, 128, 129
XFillRectangle, 105, 106, 127, 128
XFillRectangle, 105, 118, 126, 127
XIMCaretDown, 255
XIMCaretStyle, 254
XIMCaretUp, 255
XIMDontChange, 255
XIMFeedback, 253, 254
XIMForwardChar, 255
XIMForwardWord, 255
XIMHighlight, 254
XIMLineEnd, 255
XIMLineStart, 255
XIMNextLine, 255
XIMOFIC, 245
XIMPrededitArea, 239, 242, 243, 249
XIMPrededitCallbacks, 242, 243, 250
XIMPrededitCaretCallbackStruct, 254
XIMPrededitDrawCallbackStruct, 252, 253
XIMPrededitNone, 242, 243
XIMPrededitNothing, 242, 243
XIMPrededitPosition, 242, 243, 249
XIMPreviousLine, 255
XIMPrimary, 254
XIMProc, 250
XIMReverse, 254
XIMSecondary, 254
XIMStatusArea, 239, 242, 243, 249
XIMStatusCallbacks, 242, 243, 250
XIMStatusDataType, 256
XIMStatusDrawCallbackStruct, 256
XIMStatusNone, 242, 243
XIMStatusNothing, 242, 243
XIMStyle, 242
XIMStyles, 239, 242
XIMTertiary, 254
XIMText, 253
XIMUnderline, 254
XInitExtension, 337, 353
XInsertModifiermapEntry, 220
XInstallColormap, 29, 41, 62, 149, 182
XInterimAtom, 49, 51
XIntersectRegion, 310
XKeyboardControl, 212, 213
XKeyboardState, 214
XKeyCodeToKeysym, 304, 305
XKeyEvent, 164
XKeyPressEvent, 172
XKeyPressEvent, 164, 165, 304, 306
XKeyReleaseEvent, 164, 165, 304, 306
XKeysymToKeyCode, 305
XKeysymToString, 305
XKillClient, 151, 152, 285
XLastKnownRequestProcessed, 12
XLeaveWindowEvent, 165, 166, 167, 168
XLFD, 376
XlibSpecificationRelease, 3
XListExtents, 336, 337
XListFonts, 134
XListFontsWithInfo, 134, 135
XListHosts, 155
XListInstalledColormaps, 150
XListPixmapFormats, 14
XListProperties, 53, 54
XLoadFont, 132, 133
XLoadQueryFont, 129, 132, 133, 135, 339
XLocaleNotSupported, 222, 263, 264, 265
XLocaleOfFontSet, 228
XLocaleOfIM, 224, 243
XLookUpAssoc, 362
XLookUpBoth, 258
XLookUpChars, 258
XLookUpColor, 65, 69, 72
XLookUpKeySym, 258
XLookUpKeysym, 304
XLookUpNone, 258
XLookUpString, 306, 307
XLowerWindow, 39, 175, 181
XMakeAssoc, 362
XMapEvent, 177
XMappingEvent, 178
XMapRaised, 34, 175, 177, 181
XMapRequestEvent, 182
XMapSubwindows, 34, 177, 181
XMapWindow, 24, 31, 32, 33, 34, 177, 181, 182
XMaskEvent, 192
XMatchVisualInfo, 10, 315
XMaxCmapsOfScreen, 19
XMaxRequestSize, 12
XmbDrawImageString, 234, 235
XmbDrawString, 229
XmbDrawString, 229, 234
XmbDrawText, 224, 233, 234
XmbLookupString, 238, 239, 240, 241, 257, 258
XmbLookupText, 224
XmbPerCharExtents, 229
XmbResetIC, 244, 245
XmbSetWMProperties, 224, 280, 281
XmbTextEscapement, 230, 231
XmbTextExtents, 224, 229, 230, 231, 232, 234, 235
XmbTextItem, 233
XmbTextListToTextProperty, 224, 263, 264, 265
XmbTextPerCharExtents, 229, 231, 232, 234
XmbTextPropertyToTextList, 224, 264, 265, 266, 294, 295
XMinCmapsOfScreen, 19
XModifierKeymap, 219, 220, 221
XMotionEvent, 164
XMoveResizeWindow, 38, 175, 177, 181, 182, 349
XMoveWindow, 37, 175, 181, 349

391
Xlib – C Library

XrmInitialize, 294
XrmLocaleOfDatabase, 224, 295
XrmMergeDatabases, 296, 297
XrmOptionDescRec.value, 301
XrmOptionDescRec, 301
XrmOptionKind, 301
XrmoptionNoArg, 302
XrmoptionSkipArg, 301
XrmoptionSkipNArgs, 301
XrmParseCommand, 301, 302
XrmPerrnStringToQuark, 292
XrmPutFileDatabase, 224, 294
XrmPuiLineResource, 300
XrmPuiResource, 297, 299
XrmPutStringResource, 299, 300
XrmQGetResource, 297, 298
XrmQGetSearchList, 298, 299
XrmQGetSearchResource, 297, 298
XrmQPuiResource, 297, 299, 300
XrmQPutStringResource, 300
XrmQuark, 292
XrmQuarkToString, 292, 293
XrmSetDatabase, 295, 359
XrmStringToBindingQuarkList, 293, 299, 300
XrmStringToQuark, 292
XrmStringToQuarkList, 293
XrmUniqueQuark, 292
XrmValue, 293, 294, 300
XRootWindow, 13
XRootWindowOfScreen, 19
XRotateBuffers, 314
XRotateWindowProperties, 54, 55, 184
XSaveContext, 321, 322
XScreenCount, 13
XScreenNumberOfScreen, 17, 18
XScreenOfDisplay, 11
XScreenResourceString, 294
XSegment, 121, 123
XSelectInput, 187, 188
XSelectionClearEvent, 184
XSelectionEvent, 185
XSelectionRequestEvent, 185
XSendEvent, 183, 185, 194
XServerVendor, 13
XSetAccessControl, 156
XSetAfterFunction, 196
XSetArcMode, 116
XSetBackground, 110, 349
XSetClassHint, 276, 282
XSetClipMask, 115, 116
XSetClipOrigin, 115
XSetClipRectangles, 106, 108, 115, 116
XSetCloseDownMode, 20, 287
XSetCommand, 282, 283
XSetDashes, 106, 108, 111, 112
XSetErrorHandler, 196
XSetFillRule, 112
XSetFillStyle, 112
XSetFont, 114, 115
XSetFontPath, 134, 150
XSetForeground, 101, 110
XSetFunction, 110
XSetGraphicsExposures, 117, 173
XSetICFocus, 239, 244
XSetIconName, 269, 270
XSetIconSizes, 279
XSetICValues, 239, 245, 246, 247, 248
XSetInputFocus, 211, 212
XSetIOErrorHandler, 200
XSetLineAttributes, 101, 111
XSetLocaleModifiers, 223, 224, 225, 241
XSetModifierMapping, 178, 220, 221
XSetNormalHints, 355, 356
XSetPlaneMask, 110, 111
XSetPointerMapping, 178, 215
XSetRegion, 106, 309, 310
XSetRGBColormap, 357
XSetRGBColormaps, 286, 287
XSetScreenSaver, 152
XSetSelectionOwner, 20, 56, 184, 185
XSetSizeHints, 356
XSetStandardColormap, 357, 358
XSetStandardProperties, 354, 355
XSetState, 109, 110
XSetStipple, 114
XSetSubwindowMode, 116, 117
XSetTextProperty, 267, 268, 269, 283
XSetTile, 114
XSetTransientForHint, 276, 277
XSetTSOrigin, 114
XSetWindowAttributes, 24, 25, 41, 182, 187
XSetWindowBackground, 42
XSetWindowBackgroundPixmap, 42
XSetWindowBorder, 42, 43
XSetWindowBorderPixmap, 43
XSetWindowBorderWidth, 38, 39, 175, 181
XSetWindowColormap, 43, 69, 149, 182
XSetWMClientMachine, 281, 282, 283
XSetWMColormapWindows, 278
XSetWMHints, 271, 272, 280, 281, 282
XSetWMIconName, 269, 282
XSetWMName, 267, 268, 282
XSetWNormalHints, 273, 274, 281, 282, 355
XSetWMProperties, 281, 282, 354
XSetWMProtocols, 277
XSetWMSizeHints, 274, 356
XSetZoomHints, 355, 356
XShrinkRegion, 310
XSizeHints, 260, 272, 273, 274, 275, 309, 355, 356, 357
XStandardColormap, 284, 285, 286, 287, 357
XStdICCTextStyle, 264
XStoreBuffer, 313
XStoreBytes, 312, 313
XStoreColor, 74, 75, 76
XStoreColors, 74, 75, 76, 77, 287
XStoreName, 268
XStoreNamedColor, 65, 74, 77
XStoreColor, 74, 75, 76, 77, 287
XStoreColors, 74, 75, 76, 77, 287
XStoreName, 268
XStoreNamedColor, 65, 74, 77
XStringListToTextProperty, 266
XStringResourceString, 295
XStringStyle, 264
XStringToKeysym, 305
XSublmage, 316, 317, 318
XSubtractRegion, 311
XSupportsLocale, 222, 223, 224, 264, 265
XSync, 2, 20, 188
XSyncToResource, 244
XTextExtents16, 136, 137, 138
XTextExtents, 136, 137, 138
XTextltem16, 139
XTextltem, 138
XTextProperty, 263, 264, 265, 266, 267, 268, 269, 283
XTextPropertyToStringList, 266
XTextStyle, 264
XTextWidth16, 135
XTextWidth, 135
XTimeCoord, 195
XTranslateCoordinates, 48
XUndefinedCursor, 43, 44
XUngrabButton, 205
XUngrabKey, 208
XUngrabKeyboard, 20, 206, 207
XUngrabPointer, 20, 162, 165, 203
XUngrabServer, 20, 151
XUninstallColormap, 62, 69, 149, 150, 182
XUnionRectWithRegion, 311
XUnionRegion, 311
XUniqueContext, 322
XUnloadFont, 132, 133, 134
XUnmapScratch, 353
XUnmapSubwindows, 34, 35
XUnmapWindow, 34
XUnsetICFocus, 239, 244
XVaCreateNestedList, 241
XValue, 308
XVaNestedList, 241
XVendorRelease, 14
XVisibilityEvent, 179, 180
XVisualDDFromVisual, 23
XVisualInfo, 22, 314
XWindowAttributes, 46
XWindowChanges, 35, 37, 263
XWindowEvent, 2, 188, 191
XWithdrawWindow, 262
XWMGeometry, 308, 309, 358
XWMHinsics, 260, 270, 271, 272, 280, 282
XWriteBitmapFile, 319, 320
XWriteRegion, 311
XY format, 376
XYBitmap, 144, 316
XYPixmap, 144, 145, 316
X_CopyArea, 173
X_CopyPlane, 173
X_MAPWindow, 349
Y
YNegative, 308
YSorted, 116
YValue, 308
YXBanded, 116
YXSorted, 116
Z
Z format, 376
ZPixmap, 144, 145, 316
Z
ZAllocScratch, 353
ZDebug, 196
ZFlushGCCache, 345
ZRead16, 351
ZRead16Pad, 352
ZRead32, 352
ZRead, 351
ZReadPad, 352
ZReply, 342, 350, 351
ZSend, 350

394
_XSetLastRequestRead, 341
Table of Contents

Acknowledgments ........................................................................................................ iii
About This Manual ....................................................................................................... iv
Chapter 1 – Intrinsics and Widgets ........................................................................... 1
1.1. Intrinsics ................................................................................................................ 1
1.2. Languages ............................................................................................................. 1
1.3. Procedures and Macros ........................................................................................ 2
1.4. Widgets .................................................................................................................. 2
1.4.1. Core Widgets ....................................................................................................... 2
1.4.1.1. CoreClassPart Structure ............................................................................... 3
1.4.1.2. CorePart Structure ....................................................................................... 4
1.4.1.3. Core Resources ............................................................................................. 5
1.4.1.4. CorePart Default Values ................................................................................ 5
1.4.2. Composite Widgets ............................................................................................ 6
1.4.2.1. CompositeClassPart Structure .................................................................... 6
1.4.2.2. CompositePart Structure ............................................................................. 7
1.4.2.3. Composite Resources .................................................................................... 7
1.4.2.4. CompositePart Default Values ...................................................................... 7
1.4.3. Constraint Widgets ........................................................................................... 8
1.4.3.1. ConstraintClassPart Structure ...................................................................... 8
1.4.3.2. ConstraintPart Structure ............................................................................. 9
1.4.3.3. Constraint Resources .................................................................................. 9
1.5. Implementation-Specific Types ............................................................................ 9
1.6. Widget Classing .................................................................................................... 10
1.6.1. Widget Naming Conventions ............................................................................ 10
1.6.2. Widget Subclassing in Public .h Files ............................................................... 11
1.6.3. Widget Subclassing in Private .h Files .............................................................. 12
1.6.4. Widget Subclassing in .c Files ........................................................................ 14
1.6.5. Widget Class and Superclass Look Up ............................................................... 16
1.6.6. Widget Subclass Verification ............................................................................ 17
1.6.7. Superclass Chaining ........................................................................................ 18
1.6.8. Class Initialization: class_initialize and class_part_initialize Procedures .... 19
1.6.9. Initializing a Widget Class ................................................................................ 20
1.6.10. Inheritance of Superclass Operations .............................................................. 20
1.6.11. Invocation of Superclass Operations ............................................................... 21
1.6.12. Class Extension Records ............................................................................... 22
Chapter 2 – Widget Instantiation ............................................................................... 24
Acknowledgments

The design of the X11 Intrinsics was done primarily by Joel McCormack of Digital WSL. Major contributions to the design and implementation also were done by Charles Haynes, Mike Chow, and Paul Asente of Digital WSL. Additional contributors to the design and/or implementation were:

- Loretta Guarino-Reid (Digital WSL)
- Rich Hyde (Digital WSL)
- Susan Angebranndt (Digital WSL)
- Terry Weissman (Digital WSL)
- Mary Larson (Digital UEG)
- Mark Manasse (Digital SRC)
- Jim Gettys (Digital SRC)
- Ralph Swick (Project Athena and Digital ERP)
- Leo Treggiari (Digital SDT)
- Ron Newman (Project Athena)
- Mark Ackerman (Project Athena)
- Bob Scheifler (MIT LCS)

The contributors to the X10 toolkit also deserve mention. Although the X11 Intrinsics present an entirely different programming style, they borrow heavily from the implicit and explicit concepts in the X10 toolkit.

The design and implementation of the X10 Intrinsics were done by:

- Terry Weissman (Digital WSL)
- Smokey Wallace (Digital WSL)
- Phil Karlton (Digital WSL)
- Charles Haynes (Digital WSL)
- Frank Hall (HP)

The design and implementation of the X10 toolkit's sample widgets were by the above, as well as by:

- Ram Rao (Digital UEG)
- Mary Larson (Digital UEG)
- Mike Gancarz (Digital UEG)
- Kathleen Langone (Digital UEG)

These widgets provided a checklist of requirements that we had to address in the X11 intrinsics.

Thanks go to Al Mento of Digital's UEG Documentation Group for formatting and generally improving this document and to John Ousterhout of Berkeley for extensively reviewing early drafts of it.

Finally, a special thanks to Mike Chow, whose extensive performance analysis of the X10 toolkit provided the justification to redesign it entirely for X11.

Joel McCormack  
Western Software Laboratory  
Digital Equipment Corporation  

March, 1988
The current design of the Intrinsics has benefited greatly from the input of several dedicated reviewers in the membership of the X Consortium. In addition to those already mentioned, the following individuals have dedicated significant time to suggesting improvements to the Intrinsics:

- Steve Pitschke (Stellar)
- Bob Miller (HP)
- Fred Taft (HP)
- Marcel Meth (AT&T)
- Mike Collins (Digital)
- Scott McGregor (Digital)
- Julian Payne (ESS)
- Gabriel Beged-Dov (HP)
- C. Doug Blewett (AT&T)
- David Schiferl (Tektronix)
- Michael Squires (Sequent)
- Jim Fulton (MIT)
- Kerry Kimbrough (Texas Instruments)
- Phil Karlton (Digital)
- Jacques Davy (Bull)
- Glenn Widener (Tektronix)

Thanks go to each of them for the countless hours spent reviewing drafts and code.

Ralph R. Swick
External Research Group
Digital Equipment Corporation
MIT Project Athena

June, 1988

From Release 3 to Release 4, several new members joined the design team. We greatly appreciate the thoughtful comments, suggestions, lengthy discussions, and in some cases implementation code contributed by each of the following:

- Don Alecci (AT&T)
- Donna Converse (MIT)
- Nayeem Islam (Sun)
- Keith Packard (MIT)
- Richard Probst (Sun)
- Ellis Cohen (OSF)
- Clive Feather (IXI)
- Dana Laursen (HP)
- Chris Peterson (MIT)
- Larry Cable (Sun)

In Release 5, the effort to define the internationalization additions was headed by Bill McMahon of Hewlett Packard and Frank Rojas of IBM. This has been an educational process for many of us and Bill and Frank's tutelage has carried us through. Vania Joloboff of the OSF also contributed to the internationalization additions. The implementation efforts of Bill, Gabe Beged-Dov, and especially Donna Converse for this release are also gratefully acknowledged.

Ralph R. Swick

December, 1989

and

July, 1991
About This Manual

*X Toolkit Intrinsics – C Language Interface* is intended to be read by both application programmers who will use one or more of the many widget sets built with the Intrinsics and by widget programmers who will use the Intrinsics to build widgets for one of the widget sets. Not all the information in this manual, however, applies to both audiences. That is, because the application programmer is likely to use only a number of the Intrinsics functions in writing an application and because the widget programmer is likely to use many more, if not all, of the Intrinsics functions in building a widget, an attempt has been made to highlight those areas of information that are deemed to be of special interest for the application programmer. (It is assumed the widget programmer will have to be familiar with all the information.) Therefore, all entries in the table of contents that are printed in bold indicate the information that should be of special interest to an application programmer.

It is also assumed that as application programmers become more familiar with the concepts discussed in this manual they will find it more convenient to implement portions of their applications as special-purpose or custom widgets. It is possible, none the less, to use widgets without knowing how to build them.

Conventions Used in this Manual

This document uses the following conventions:

- Global symbols are printed in this special font. These can be either function names, symbols defined in include files, data types, or structure names. Arguments to functions, procedures, or macros are printed in *italics*.

- Each function is introduced by a general discussion that distinguishes it from other functions. The function declaration itself follows, and each argument is specifically explained. General discussion of the function, if any is required, follows the arguments.

- To eliminate any ambiguity between those arguments that you pass and those that a function returns to you, the explanations for all arguments that you pass start with the word *specifies* or, in the case of multiple arguments, the word *specify*. The explanations for all arguments that are returned to you start with the word *returns* or, in the case of multiple arguments, the word *return*. 
Chapter 1
Intrinsics and Widgets

The Intrinsics are a programming library tailored to the special requirements of user interface construction within a network window system, specifically the X Window System. The Intrinsics and a widget set make up an X Toolkit.

1.1. Intrinsics

The Intrinsics provide the base mechanism necessary to build a wide variety of interoperating widget sets and application environments. The Intrinsics are a layer on top of Xlib, the C Library X Interface. They extend the fundamental abstractions provided by the X Window System while still remaining independent of any particular user interface policy or style.

The Intrinsics use object-oriented programming techniques to supply a consistent architecture for constructing and composing user interface components, known as widgets. This allows programmers to extend a widget set in new ways, either by deriving new widgets from existing ones (subclassing), or by writing entirely new widgets following the established conventions.

When the Intrinsics were first conceived, the root of the object hierarchy was a widget class named Core. In release 4 of the Intrinsics, three nonwidget superclasses were added above Core. These superclasses are described in Chapter 12. The name of the class now at the root of the Intrinsics class hierarchy is Object. The remainder of this specification refers uniformly to widgets and Core as if they were the base class for all Intrinsics operations. The argument descriptions for each Intrinsics procedure and Chapter 12 describe which operations are defined for the nonwidget superclasses of Core. The reader may determine by context whether a specific reference to widget actually means widget or object.

1.2. Languages

The Intrinsics are intended to be used for two programming purposes. Programmers writing widgets will be using most of the facilities provided by the Intrinsics to construct user interface components from the simple, such as buttons and scrollbars, to the complex, such as control panels and property sheets. Application programmers will use a much smaller subset of the Intrinsics procedures in combination with one or more sets of widgets to construct and present complete user interfaces on an X display. The Intrinsics programming interfaces primarily intended for application use are designed to be callable from most procedural programming languages. Therefore, most arguments are passed by reference rather than by value. The interfaces primarily intended for widget programmers are expected to be used principally from the C language. In these cases, the usual C programming conventions apply. In this specification, the term client refers to any module, widget, or application that calls an Intrinsics procedure.

Applications that use the Intrinsics mechanisms must include the header files <X11/Intrinsic.h> and <X11/StringDefs.h>, or their equivalent, and they may also include <X11/Atoms.h> and <X11/Shell.h>. In addition, widget implementations should include <X11/IntrinsicP.h> instead of <X11/Intrinsic.h>.

The applications must also include the additional header files for each widget class that they are to use (for example, <X11/Xaw/Label.h> or <X11/Xaw/Scrollbar.h>). On a POSIX-based system, the Intrinsics object library file is named libXt.a and is usually referenced as -lXt when linking the application.
1.3. Procedures and Macros

All functions defined in this specification except those specified below may be implemented as C macros with arguments. C applications may use "#undef" to remove a macro definition and ensure that the actual function is referenced. Any such macro will expand to a single expression which has the same precedence as a function call and that evaluates each of its arguments exactly once, fully protected by parentheses, so that arbitrary expressions may be used as arguments.

The following symbols are macros that do not have function equivalents and that may expand their arguments in a manner other than that described above: XtCheckSubclass, XtNew, XtNumber, XtOffsetOf, XtOffset, and XtSetArg.

1.4. Widgets

The fundamental abstraction and data type of the X Toolkit is the widget, which is a combination of an X window and its associated input and display semantics and which is dynamically allocated and contains state information. Some widgets display information (for example, text or graphics), and others are merely containers for other widgets (for example, a menu box). Some widgets are output-only and do not react to pointer or keyboard input, and others change their display in response to input and can invoke functions that an application has attached to them.

Every widget belongs to exactly one widget class, which is statically allocated and initialized and which contains the operations allowable on widgets of that class. Logically, a widget class is the procedures and data associated with all widgets belonging to that class. These procedures and data can be inherited by subclasses. Physically, a widget class is a pointer to a structure. The contents of this structure are constant for all widgets of the widget class but will vary from class to class. (Here, "constant" means the class structure is initialized at compile time and never changed, except for a one-time class initialization and in-place compilation of resource lists, which takes place when the first widget of the class or subclass is created.) For further information, see Section 2.5.

The distribution of the declarations and code for a new widget class among a public .h file for application programmer use, a private .h file for widget program use, and the implementation .c file is described in Section 1.6. The predefined widget classes adhere to these conventions.

A widget instance is composed of two parts:
- A data structure which contains instance-specific values.
- A class structure which contains information that is applicable to all widgets of that class.

Much of the input/output of a widget (for example, fonts, colors, sizes, border widths, and so on) is customizable by users.

This chapter discusses the base widget classes, Core, Composite, and Constraint, and ends with a discussion of widget classing.

1.4.1. Core Widgets

The Core widget class contains the definitions of fields common to all widgets. All widgets classes are subclasses of the Core class, which is defined by the CoreClassPart and CorePart structures.
1.4.1.1. CoreClassPart Structure

All widget classes contain the fields defined in the CoreClassPart structure.

typedef struct {
    WidgetClass superclass;                        See Section 1.6
    String class_name;                              See Chapter 9
    Cardinal widget_size;                           See Section 1.6
    XtProc class_initialize;                        See Section 1.6
    XtWidgetClassProc class_part_initialize;       See Section 1.6
    XtEnum class_initialized;                       See Section 1.6
    XtInitProc initialize;                          See Section 2.5
    XtArgsProc initialize_hook;                    See Section 2.5
    XtRealizeProc realize;                          See Section 2.6
    XtActionList actions;                           See Section 10
    Cardinal num_actions;                          See Chapter 10
    XtResourceList resources;                      See Chapter 9
    Cardinal num_resources;                        See Chapter 9
    XrmClass xrm_class;                             Private to resource manager
    Boolean compress_motion;                        See Section 7.9
    XtEnum compress_exposure;                       See Section 7.9
    Boolean compress_enterleave;                   See Section 7.9
    Boolean visible_interest;                      See Section 7.10
    XtWidgetProc destroy;                           See Section 2.8
    XtWidgetProc resize;                            See Chapter 6
    XtExposeProc expose;                            See Section 7.10
    XSetValuesFunc set_values;                     See Section 9.7
    XtArgsFunc set_values_hook;                    See Section 9.7
    XtAlmostProc set_values_almost;                See Section 9.7
    XtArgsProc get_values_hook;                    See Section 9.7
    XtAcceptFocusProc accept_focus;                See Section 7.3
    XVersionType version;                           See Section 1.6
    XtPointer callback_private;                    Private to callbacks
    String tm_table;                                See Chapter 10
    XtGeometryHandler query_geometry;              See Chapter 6
    XtStringProc display_accelerator;              See Chapter 10
    XtPointer extension;                           See Section 1.6
} CoreClassPart;

All widget classes have the Core class fields as their first component. The prototypical WidgetClass and CoreWidgetClass are defined with only this set of fields.

typedef struct {
    CoreClassPart core_class;
} WidgetClassRec, *WidgetClass, CoreClassRec, *CoreWidgetClass;

Various routines can cast widget class pointers, as needed, to specific widget class types. The single occurrences of the class record and pointer for creating instances of Core are

In IntrinsicP.h:
extern WidgetClassRec widgetClassRec;
define coreClassRec widgetClassRec

In Intrinsic.h:
extern WidgetClass widgetClass, coreWidgetClass:
The opaque types Widget and WidgetClass and the opaque variable widgetClass are defined for generic actions on widgets. In order to make these types opaque and ensure that the compiler does not allow applications to access private data, the Intrinsics use incomplete structure definitions in Intrinsic.h:

typedef struct _WidgetClassRec *WidgetClass, *CoreWidgetClass;

1.4.1.2. CorePart Structure

All widget instances contain the fields defined in the CorePart structure.

typedef struct _CorePart {
    Widget self;
    WidgetClass widget_class;  See Section 1.6
    Widget parent;  See Section 2.5
    Boolean being_destroyed;  See Section 2.8
    XtCallbackList destroy_callbacks;  See Section 2.8
    XtPointer constraints;  See Section 3.6
    Position x;  See Chapter 6
    Position y;  See Chapter 6
    Dimension width;  See Chapter 6
    Dimension height;  See Chapter 6
    Dimension border_width;  See Chapter 6
    Boolean managed;  See Chapter 3
    Boolean sensitive;  See Section 7.7
    Boolean ancestor_sensitive;  See Section 7.7
    XtTranslations accelerators;  See Chapter 10
    Pixel border_pixel;  See Section 2.6
    Pixmap border_pixmap;  See Section 2.6
    WidgetList popup_list;  See Chapter 5
    Cardinal num_popups;  See Chapter 5
    String name;  See Chapter 9
    Screen *screen;  See Section 2.6
    Colormap colormap;  See Section 2.6
    Window window;  See Section 2.6
    Cardinal depth;  See Section 2.6
    Pixel background_pixel;  See Section 2.6
    Pixmap background_pixmap;  See Section 2.6
    Boolean visible;  See Section 7.10
    Boolean mapped_when_managed;  See Chapter 3
} CorePart;

All widget instances have the Core fields as their first component. The prototypical type Widget is defined with only this set of fields.

typedef struct {
    CorePart core;
} WidgetRec, *Widget, CoreRec, *CoreWidget;

Various routines can cast widget pointers, as needed, to specific widget types.

In order to make these types opaque and ensure that the compiler does not allow applications to access private data, the Intrinsics use incomplete structure definitions in Intrinsic.h.

typedef struct _WidgetRec *Widget, *CoreWidget;
1.4.1.3. Core Resources

The resource names, classes, and representation types specified in the coreClassRec resource list are

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerators</td>
<td>XtCAccelerators</td>
<td>XtRAcceleratorTable</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>XtCBackground</td>
<td>XtRPixel</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>XtCPixmap</td>
<td>XtRPixmap</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>XtCBorderColor</td>
<td>XtRPixel</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>XtCPixmap</td>
<td>XtRPixmap</td>
</tr>
<tr>
<td>XtN colormap</td>
<td>XtCColormap</td>
<td>XtRColormap</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>XtCDepth</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtN mappedWhenManaged</td>
<td>XtCMappedWhenManaged</td>
<td>XtRBoolean</td>
</tr>
<tr>
<td>XtNscreen</td>
<td>XtCScreen</td>
<td>XtRScreen</td>
</tr>
<tr>
<td>XtN translations</td>
<td>XtCTranslations</td>
<td>XtRTranslationTable</td>
</tr>
</tbody>
</table>

Additional resources are defined for all widgets via the objectClassRec and rectObjClassRec resource lists; see Sections 12.2 and 12.3 for details.

1.4.1.4. CorePart Default Values

The default values for the Core fields, which are filled in from the resource lists and by the initialize procedures, are

<table>
<thead>
<tr>
<th>Field</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>Address of the widget structure (may not be changed).</td>
</tr>
<tr>
<td>widget_class</td>
<td><em>widget_class</em> argument to <em>XtCreateWidget</em> (may not be changed).</td>
</tr>
<tr>
<td>parent</td>
<td><em>parent</em> argument to <em>XtCreateWidget</em> (may not be changed).</td>
</tr>
<tr>
<td>being_destroyed</td>
<td>Parent's <em>being_destroyed</em> value.</td>
</tr>
<tr>
<td>destroy_callbacks</td>
<td>NULL</td>
</tr>
<tr>
<td>constraints</td>
<td>NULL</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
</tr>
<tr>
<td>width</td>
<td>0</td>
</tr>
<tr>
<td>height</td>
<td>0</td>
</tr>
<tr>
<td>border_width</td>
<td>1</td>
</tr>
<tr>
<td>managed</td>
<td>False</td>
</tr>
<tr>
<td>sensitive</td>
<td>True</td>
</tr>
<tr>
<td>ancestor_sensitive</td>
<td>logical AND of parent's <em>sensitive</em> and <em>ancestor_sensitive</em> values.</td>
</tr>
<tr>
<td>accelerators</td>
<td>NULL</td>
</tr>
<tr>
<td>border_pixel</td>
<td><em>XtDefaultForeground</em></td>
</tr>
<tr>
<td>border pixmap</td>
<td><em>XtUnspecifiedPixmap</em></td>
</tr>
<tr>
<td>popup_list</td>
<td>NULL</td>
</tr>
<tr>
<td>num_popups</td>
<td>0</td>
</tr>
<tr>
<td>name</td>
<td><em>name</em> argument to <em>XtCreateWidget</em> (may not be changed).</td>
</tr>
<tr>
<td>screen</td>
<td>Parent's <em>screen</em>; top-level widget gets screen from display specifier (may not be changed).</td>
</tr>
<tr>
<td>colormap</td>
<td>Parent's <em>colormap</em> value.</td>
</tr>
<tr>
<td>window</td>
<td>NULL</td>
</tr>
<tr>
<td>depth</td>
<td>Parent's <em>depth</em>; top-level widget gets root window depth.</td>
</tr>
</tbody>
</table>
X Toolkit Intrinsics

background_pixel       XtDefaultBackground
background_pixmap     XtUnspecifiedPixmap
visible               True
mapped_when_managed   True

XtUnspecifiedPixmap is a symbolic constant guaranteed to be unequal to any valid Pixmap id, None, and ParentRelative.

1.4.2. Composite Widgets

The Composite widget class is a subclass of the Core widget class (see Chapter 3). Composite widgets are intended to be containers for other widgets. The additional data used by composite widgets are defined by the CompositeClassPart and CompositePart structures.

1.4.2.1. CompositeClassPart Structure

In addition to the Core class fields, widgets of the Composite class have the following class fields.

```c
typedef struct {
    XtGeometryHandler geometry_manager;       See Chapter 6
    XtWidgetProc change_managed;              See Chapter 3
    XtWidgetProc insert_child;               See Chapter 3
    XtWidgetProc delete_child;               See Chapter 3
    XtPointer extension;                     See Section 1.6
} CompositeClassPart;
```

The extension record defined for CompositeClassPart with record_type equal to NULLQUARK is CompositeClassExtensionRec.

```c
typedef struct {
    XtPointer next_extension;               See Section 1.6.12
    XrmQuark record_type;                   See Section 1.6.12
    long version;                           See Section 1.6.12
    Cardinal record_size;                   See Section 1.6.12
    Boolean accepts_objects;                See Chapter 3
} CompositeClassExtensionRec, *CompositeClassExtension;
```

Composite classes have the Composite class fields immediately following the Core class fields.

```c
typedef struct {
    CoreClassPart core_class;
    CompositeClassPart composite_class;
} CompositeClassRec, *CompositeWidgetClass;
```

The single occurrences of the class record and pointer for creating instances of Composite are in IntrinsicP.h:

```
extern CompositeClassRec compositeClassRec;
```

In Intrinsic.h:

```
extern WidgetClass compositeWidgetClass;
```

The opaque types CompositeWidget and CompositeWidgetClass and the opaque variable compositeWidgetClass are defined for generic operations on widgets whose class is Composite or a subclass of Composite. The symbolic constant for the CompositeClassExtension
version identifier is \texttt{XtCompositeExtensionVersion} (see Section 1.6.12). \texttt{Intrinsic.h} uses an incomplete structure definition to ensure that the compiler catches attempts to access private data.

typedef struct _CompositeClassRec *CompositeWidgetClass;

1.4.2.2. CompositePart Structure

In addition to the Core instance fields, widgets of the Composite class have the following instance fields defined in the \texttt{CompositePart} structure.

typedef struct {
    WidgetList children; See Chapter 3
    Cardinal num_children; See Chapter 3
    Cardinal num_slots; See Chapter 3
    XtOrderProc insert_position; See Section 3.2
} CompositePart;

Composite widgets have the Composite instance fields immediately following the Core instance fields.

typedef struct {
    CorePart core;
    CompositePart composite;
} CompositeRec, *CompositeWidget;

\texttt{Intrinsic.h} uses an incomplete structure definition to ensure that the compiler catches attempts to access private data.

typedef struct _CompositeRec *CompositeWidget;

1.4.2.3. Composite Resources

The resource names, classes, and representation types that are specified in the \texttt{compositeClassRec} resource list are

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNchildren</td>
<td>XtCReadOnly</td>
<td>XtRWidgetList</td>
</tr>
<tr>
<td>XtNinsertPosition</td>
<td>XtCInsertPosition</td>
<td>XtRFunction</td>
</tr>
<tr>
<td>XtNnumChildren</td>
<td>XtCReadOnly</td>
<td>XtRCardinal</td>
</tr>
</tbody>
</table>

1.4.2.4. CompositePart Default Values

The default values for the Composite fields, which are filled in from the Composite resource list and by the Composite initialize procedure, are

<table>
<thead>
<tr>
<th>Field</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>NULL</td>
</tr>
<tr>
<td>num_children</td>
<td>0</td>
</tr>
<tr>
<td>num_slots</td>
<td>0</td>
</tr>
<tr>
<td>insert_position</td>
<td>Internal function to insert at end</td>
</tr>
</tbody>
</table>
The children, num_children, and insert_position fields are declared as resources; XtNinsertPosition is a settable resource, XtNchildren and XtNnumChildren may be read by any client but should only be modified by the composite widget class procedures.

1.4.3. Constraint Widgets

The Constraint widget class is a subclass of the Composite widget class (see Section 3.6). Constraint widgets maintain additional state data for each child; for example, client-defined constraints on the child’s geometry. The additional data used by constraint widgets are defined by the ConstraintClassPart and ConstraintPart structures.

1.4.3.1. ConstraintClassPart Structure

In addition to the Core and Composite class fields, widgets of the Constraint class have the following class fields.

typedef struct {
    XtResourceList resources; See Chapter 9
    Cardinal num_resources; See Chapter 9
    Cardinal constraint_size; See Section 3.6
    XtInitProc initialize; See Section 3.6
    XtWidgetProc destroy; See Section 3.6
    XtSetValuesFunc set_values; See Section 9.7.2
    XtPointer extension; See Section 1.6
} ConstraintClassPart;

The extension record defined for ConstraintClassPart with record_type equal to NULLQUARK is ConstraintClassExtensionRec.

typedef struct {
    XtPointer next_extension; See Section 1.6.12
    XrmQuark record_type; See Section 1.6.12
    long version; See Section 1.6.12
    Cardinal record_size; See Section 1.6.12
    XtArgsProc get_values_hook; See Section 9.7.1
} ConstraintClassExtensionRec, *ConstraintClassExtension;

Constraint classes have the Constraint class fields immediately following the Composite class fields.

typedef struct _ConstraintClassRec {
    CoreClassPart core_class;
    CompositeClassPart composite_class;
    ConstraintClassPart constraint_class;
} ConstraintClassRec, *ConstraintWidgetClass;

The single occurrences of the class record and pointer for creating instances of Constraint are

In IntrinsicP.h:
extern ConstraintClassRec constraintClassRec;

In Intrinsic.h:
extern WidgetClass constraintWidgetClass;

The opaque types ConstraintWidget and ConstraintWidgetClass and the opaque variable constraintWidgetClass are defined for generic operations on widgets whose class is Constraint or a subclass of Constraint. The symbolic constant for the ConstraintClassExtension
version identifier is XtConstraintExtensionVersion (see Section 1.6.12). Intrinsic.h uses an incomplete structure definition to ensure that the compiler catches attempts to access private data.

typedef struct _ConstraintClassRec *ConstraintWidgetClass;

1.4.3.2. ConstraintPart Structure
In addition to the Core and Composite instance fields, widgets of the Constraint class have the following unused instance fields defined in the ConstraintPart structure

typedef struct {
    int empty;
} ConstraintPart;

Constraint widgets have the Constraint instance fields immediately following the Composite instance fields.

typedef struct {
    CorePart core;
    CompositePart composite;
    ConstraintPart constraint;
} ConstraintRec, *ConstraintWidget;

Intrinsic.h uses an incomplete structure definition to ensure that the compiler catches attempts to access private data.

typedef struct _ConstraintRec *ConstraintWidget;

1.4.3.3. Constraint Resources
The constraintClassRec core_class and constraint_class resources fields are NULL and the num_resources fields are zero; no additional resources beyond those declared by the superclasses are defined for Constraint.

1.5. Implementation-specific Types
To increase the portability of widget and application source code between different system environments, the Intrinsics define several types whose precise representation is explicitly dependent upon, and chosen by, each individual implementation of the Intrinsics.

These implementation-defined types are

- **Boolean**
  A datum that contains a zero or nonzero value. Unless explicitly stated, clients should not assume that the nonzero value is equal to the symbolic value True.

- **Cardinal**
  An unsigned integer datum with a minimum range of $[0..2^{16}-1]$.

- **Dimension**
  An unsigned integer datum with a minimum range of $[0..2^{16}-1]$.

- **Position**
  A signed integer datum with a minimum range of $[-2^{15}..2^{15}-1]$.

- **XtPointer**
  A datum large enough to contain the largest of a char*, int*, function pointer, structure pointer, or long value. A pointer to any type or function, or a long value may be converted to an XtPointer and back again and the result will compare equal to the original value. In ANSI C environments it is expected that XtPointer will be defined as void*.

- **XtArgVal**
  A datum large enough to contain an XtPointer, Cardinal, Dimension, or Position value.

- **XtEnum**
  An integer datum large enough to encode at least 128 distinct values, two of which are the symbolic values True and False. The symbolic values TRUE and FALSE are also defined to be equal to True and False, respectively.
In addition to these specific types, the precise order of the fields within the structure declarations for any of the instance part records ObjectPart, RectObjPart, CorePart, CompositePart, ShellPart, WMShellPart, TopLevelShellPart, and ApplicationShellPart is implementation-defined. These structures may also have additional private fields internal to the implementation. The ObjectPart, RectObjPart, and CorePart structures must be defined so that any member with the same name appears at the same offset in ObjectRec, RectObjRec and CoreRec (WidgetRec). No other relations between the offsets of any two fields may be assumed.

1.6. Widget Classing

The widget_class field of a widget points to its widget class structure, which contains information that is constant across all widgets of that class. As a consequence, widgets usually do not implement directly callable procedures; rather, they implement procedures, called methods, that are available through their widget class structure. These methods are invoked by generic procedures that envelop common actions around the methods implemented by the widget class. Such procedures are applicable to all widgets of that class and also to widgets whose classes are subclasses of that class.

All widget classes are a subclass of Core and can be subclassed further. Subclassing reduces the amount of code and declarations necessary to make a new widget class that is similar to an existing class. For example, you do not have to describe every resource your widget uses in an XtResourceList. Instead, you describe only the resources your widget has that its superclass does not. Subclasses usually inherit many of their superclasses' procedures (for example, the expose procedure or geometry handler).

Subclassing, however, can be taken too far. If you create a subclass that inherits none of the procedures of its superclass, you should consider whether you have chosen the most appropriate superclass.

To make good use of subclassing, widget declarations and naming conventions are highly stylized. A widget consists of three files:

- A public .h file, used by client widgets or applications.
- A private .h file, used by widgets whose classes are subclasses of the widget class.
- A .c file, which implements the widget.

1.6.1. Widget Naming Conventions

The Intrinsics provide a vehicle by which programmers can create new widgets and organize a collection of widgets into an application. To ensure that applications need not deal with as many styles of capitalization and spelling as the number of widget classes it uses, the following guidelines should be followed when writing new widgets:

- Use the X library naming conventions that are applicable. For example, a record component name is all lower case and uses underscores (_) for compound words (for example, background_pixmap). Type and procedure names start with upper case and use capitalization for compound words (for example, ArgList or XtSetValues).
- A resource name is spelled identically to the field name except that compound names use capitalization rather than underscore. To let the compiler catch spelling errors, each resource name should have a symbolic identifier prefixed with "XtN". For example, the background_pixmap field has the corresponding identifier XtNbackgroundPixmap, which is defined as the string "backgroundPixmap". Many predefined names are listed in <X11/StringDefs.h>. Before you invent a new name, you should make sure there is not already a name that you can use.
A resource class string starts with a capital letter and uses capitalization for compound names (for example, "BorderWidth"). Each resource class string should have a symbolic identifier prefixed with "XtC" (for example, XtCBorderWidth). Many predefined classes are listed in <X11/StringDefs.h>.

A resource representation string is spelled identically to the type name (for example, "TranslationTable"). Each representation string should have a symbolic identifier prefixed with "XtR" (for example, XtRTranslationTable). Many predefined representation types are listed in <X11/StringDefs.h>.

New widget classes start with a capital and use upper case for compound words. Given a new class name AbcXyz, you should derive several names:
- Additional widget instance structure part name AbcXyzPart.
- Complete widget instance structure names AbcXyzRec and _AbcXyzRec.
- Widget instance structure pointer type name AbcXyzWidget.
- Additional class structure part name AbcXyzClassPart.
- Complete class structure names AbcXyzClassRec and _AbcXyzClassRec.
- Class structure pointer type name AbcXyzWidgetClass.
- Class structure variable abcXyzClassRec.
- Class structure pointer variable abcXyzWidgetClass.

Action procedures available to translation specifications should follow the same naming conventions as procedures. That is, they start with a capital letter, and compound names use upper case (for example, "Highlight" and "NotifyClient").

The symbolic identifiers XtN..., XtC... and XtR... may be implemented as macros, as global symbols, or as a mixture of the two. The (implicit) type of the identifier is String. The pointer value itself is not significant; clients must not assume that inequality of two identifiers implies inequality of the resource name, class, or representation string. Clients should also note that although global symbols permit savings in literal storage in some environments, they also introduce the possibility of multiple definition conflicts when applications attempt to use independently developed widgets simultaneously.

1.6.2. Widget Subclassing in Public .h Files
The public .h file for a widget class is imported by clients and contains
- A reference to the public .h file for the superclass.
- Symbolic identifiers for the names and classes of the new resources that this widget adds to its superclass. The definitions should have a single space between the definition name and the value and no trailing space or comment in order to reduce the possibility of compiler warnings from similar declarations in multiple classes.
- Type declarations for any new resource data types defined by the class.
- The class record pointer variable used to create widget instances.
- The C type that corresponds to widget instances of this class.
- Entry points for new class methods.

For example, the following is the public .h file for a possible implementation of a Label widget:

```c
#ifndef LABEL_H
#define LABEL_H

/* New resources */
#define XtNjustify "justify"

11```
#define XtNforeground "foreground"
#define XtNlabel "label"
#define XtNfont "font"
#define XtNinternalWidth "internalWidth"
#define XtNinternalHeight "internalHeight"

/* Class record pointer */
extern WidgetClass labelWidgetClass;

/* C Widget type definition */
typedef struct _LabelRec *LabelWidget;

/* New class method entry points */
extern void LabelSetText();
    /* Widget w */
    /* String text */

text String LabelGetText();
    /* Widget w */

#endif LABEL_H

The conditional inclusion of the text allows the application to include header files for different
widgets without being concerned that they already may be included as a superclass of another
widget.

To accommodate operating systems with file name length restrictions, the name of the public .h
file is the first ten characters of the widget class. For example, the public .h file for the Con¬
straint widget class is Constraint.h.

1.6.3. Widget Subclassing in Private .h Files
The private .h file for a widget is imported by widget classes that are subclasses of the widget
and contains
- A reference to the public .h file for the class.
- A reference to the private .h file for the superclass.
- Symbolic identifiers for any new resource representation types defined by the class. The
definitions should have a single space between the definition name and the value and no
trailing space or comment.
- A structure part definition for the new fields that the widget instance adds to its
superclass’s widget structure.
- The complete widget instance structure definition for this widget.
- A structure part definition for the new fields that this widget class adds to its superclass’s
constraint structure if the widget class is a subclass of Constraint.
- The complete constraint structure definition if the widget class is a subclass of Con¬
straint.
- Type definitions for any new procedure types used by class methods declared in the wid¬
gen class part
- A structure part definition for the new fields that this widget class adds to its superclass’s
widget class structure.
- The complete widget class structure definition for this widget.
• The complete widget class extension structure definition for this widget, if any.
• The symbolic constant identifying the class extension version, if any.
• The name of the global class structure variable containing the generic class structure for this class.
• An inherit constant for each new procedure in the widget class part structure.

For example, the following is the private .h file for a possible Label widget:

```c
#ifndef LABELP_H
#define LABELP_H

#include <X11/Label.h>

/* New representation types used by the Label widget */
#define XtRJustify "Justify"

/* New fields for the Label widget record */
typedef struct {
    /* Settable resources */
    Pixel foreground;
    XFontStruct *font;
    String label; /* text to display */
    XtJustify justify;
    Dimension internal_width; /* # pixels horizontal border */
    Dimension internal_height; /* # pixels vertical border */

    /* Data derived from resources */
    GC normal_GC;
    GC gray_GC;
   Pixmap gray_pixmap;
    Position label_x;
    Position label_y;
    Dimension label_width;
    Dimension label_height;
    Cardinal label_len;
    Boolean display_sensitive;
} LabelPart;

/* Full instance record declaration */
typedef struct _LabelRec {
    CorePart core;
    LabelPart label;
} LabelRec;

/* Types for Label class methods */
typedef void (*LabelSetTextProc)();
    /* Widget w */
    /* String text */

typedef String (*LabelGetTextProc)();
    /* Widget w */

/* New fields for the Label widget class record */
typedef struct {
```
X Toolkit Intrinsics

X Toolkit Intrinsics

1.6.4. Widget Subclassing in .c Files

The .c file for a widget contains the structure initializer for the class record variable, which contains the following parts:

- Class information (for example, superclass, class_name, widget_size, class_initialize, and class_initied).
- Data constants (for example, resources and num_resources, actions and num_actions, visible_interest, compress_motion, compress_exposure, and version)
- Widget operations (for example, initialize, realize, destroy, resize, expose, set_values, accept_focus, and any new operations specific to the widget).

The superclass field points to the superclass global class record, declared in the superclass private .h file. For direct subclasses of the generic core widget, superclass should be initialized to the address of the widgetClassRec structure. The superclass is used for class chaining operations and for inheriting or enveloping a superclass’s operations (see Sections 1.6.7, 1.6.9, and 1.6.10).

The class_name field contains the text name for this class, which is used by the resource manager. For example, the Label widget has the string “Label”. More than one widget class can share the same text class name. This string must be permanently allocated prior to or during the execution of the class initialization procedure and must not be subsequently deallocated.

The widget_size field is the size of the corresponding widget instance structure (not the size of the class structure).

The version field indicates the toolkit implementation version number and is used for runtime consistency checking of the X Toolkit and widgets in an application. Widget writers must set it to the implementation-defined symbolic value XtVersion in the widget class structure initialization. Those widget writers who believe that their widget binaries are compatible with other implementations of the Intrinsics can put the special value XtVersionDontCheck in the version field to disable version checking for those widgets. If a widget needs to compile alternative code for different revisions of the Intrinsics interface definition, it may use the symbol XtSpecificationRelease, as described in Chapter 13. Use of XtVersion allows the Intrinsics
implementation to recognize widget binaries that were compiled with older implementations. The 
extension field is for future forward compatibility. If the widget programmer adds fields to 
class parts, all subclass structures change, requiring complete recompilation. To allow 
clients to avoid recompilation, the extension field at the end of each class part can point to a 
record that contains any additional class information required.

All other fields are described in their respective sections.
The .c file also contains the declaration of the global class structure pointer variable used to 
create instances of the class. The following is an abbreviated version of the .c file for a Label 
widget. The resources table is described in Chapter 9.

/* Resources specific to Label */
static XtResource resources[] = {
    {XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),
     XtOffset(LabelWidget, label.foreground), XtRString,
     XtDefaultForeground},
    {XtNfont, XtCFont, XtRFontStruct, sizeof(XFontStruct *),
     XtOffset(LabelWidget, label.font), XtRString,
     XtDefaultFont},
    {XtNlabel, XtCLabel, XtRString, sizeof(String),
     XtOffset(LabelWidget, label.label), XtRString, NULL},

    /* Forward declarations of procedures */
    static void ClassInitialize();
    static void Initialize();
    static void Realize();
    static void SetText();
    static void GetText();

    /* Class record constant */
    LabelClassRec labelClassRec = {
        /* core_class fields */
        /* superclass */ (WidgetClass)&coreClassRec,
        /* class_name */ "Label",
        /* widget_size */ sizeof(LabelRec),
        /* class_initialize */ ClassInitialize,
        /* class_part_initialize */ NULL,
        /* class_inited */ False,
        /* initialize */ Initialize,
        /* initialize_hook */ NULL,
        /* realize */ Realize,
        /* actions */ NULL,
        /* num_actions */ 0,
        /* resources */ resources,
        /* num_resources */ XtNumber(resources),
    }
};
/* xrm_class */
/* compress_motion */
/* compress_exposure */
/* compress_enterleave */
/* visible_interest */
/* destroy */
/* resize */
/* expose */
/* set_values */
/* set_values_hook */
/* set_values_almost */
/* get_values_hook */
/* accept_focus */
/* version */
/* callback_offsets */
/* tm_table */
/* query_geometry */
/* display_accelerator */
/* extension */

void LabelSetText(w, text)
    Widget w;
    String text;
    { Label WidgetGass lwc = (Label WidgetGass)XtGass(w);
      XtCheckSubclass(w, labelWidgetGass, NULL);
      *(lwc->label_class.set_text)(w, text); }

1.6.5. Widget Class and Superclass Look Up
To obtain the class of a widget, use XtClass.
WidgetClass XtClass(w)
    Widget w;

w specifies the widget. Must be of class Object or any subclass thereof.
The `XtClass` function returns a pointer to the widget's class structure.

To obtain the superclass of a widget, use `XtSuperclass`.

```c
WidgetClass XtSuperclass(w)
    Widget w;
```

`w` Specifies the widget. Must be of class Object or any subclass thereof.

The `XtSuperclass` function returns a pointer to the widget's superclass class structure.

### 1.6.6. Widget Subclass Verification

To check the subclass to which a widget belongs, use `XtIsSubclass`.

```c
Boolean XtIsSubclass(w, widget_class)
    Widget w;
    WidgetClass widget_class;
```

`w` Specifies the widget or object instance whose class is to be checked. Must be of class Object or any subclass thereof.

`widget_class` Specifies the widget class for which to test. Must be objectClass or any subclass thereof.

The `XtIsSubclass` function returns True if the class of the specified widget is equal to or is a subclass of the specified class. The widget's class can be any number of subclasses down the chain and need not be an immediate subclass of the specified class. Composite widgets that need to restrict the class of the items they contain can use `XtIsSubclass` to find out if a widget belongs to the desired class of objects.

To test if a given widget belongs to a subclass of an Intrinsics-defined class, the Intrinsics define macros or functions equivalent to `XtIsSubclass` for each of the built-in classes. These procedures are `XtIsObject`, `XtIsRectObj`, `XtIsWidget`, `XtIsComposite`, `XtIsConstraint`, `XtIsShell`, `XtIsOverrideShell`, `XtIsWMShell`, `XtIsVendorShell`, `XtIsTransientShell`, `XtIsTopLevelShell` and `XtIsApplicationShell`.

All these macros and functions have the same argument description.

```c
Boolean XtIs<class>(w)
    Widget w;
```

`w` Specifies the widget or object instance whose class is to be checked. Must be of class Object or any subclass thereof.

These procedures may be faster than calling `XtIsSubclass` directly for the built-in classes.

To check a widget's class and to generate a debugging error message, use `XtCheckSubclass`, defined in `<X11/IntrinsicP.h>`:

```c
void XtCheckSubclass(w, widget_class, message)
    Widget w;
    WidgetClass widget_class;
    String message;
```

`w` Specifies the widget or object whose class is to be checked. Must be of class Object or any subclass thereof.

`widget_class` Specifies the widget class for which to test. Must be objectClass or any subclass thereof.

`message` Specifies the message to be used.
The **XtCheckSubclass** macro determines if the class of the specified widget is equal to or is a subclass of the specified class. The widget’s class can be any number of subclasses down the chain and need not be an immediate subclass of the specified class. If the specified widget’s class is not a subclass, **XtCheckSubclass** constructs an error message from the supplied message, the widget’s actual class, and the expected class and calls **XtErrorMsg**. **XtCheckSubclass** should be used at the entry point of exported routines to ensure that the client has passed in a valid widget class for the exported operation.

**XtCheckSubclass** is only executed when the module has been compiled with the compiler symbol **DEBUG** defined; otherwise, it is defined as the empty string and generates no code.

### 1.6.7. Superclass Chaining

While most fields in a widget class structure are self-contained, some fields are linked to their corresponding fields in their superclass structures. With a linked field, the Intrinsics access the field’s value only after accessing its corresponding superclass value (called downward superclass chaining) or before accessing its corresponding superclass value (called upward superclass chaining). The self-contained fields are:

In all widget classes:

- `class_name`
- `class_initialize`
- `widget_size`
- `realize`
- `visible_interest`
- `resize`
- `expose`
- `accept_focus`
- `compress_motion`
- `compress_exposure`
- `compress_enterleave`
- `set_values_almost`
- `tm_table`
- `version`

In Composite widget classes:

- `geometry_manager`
- `change_managed`
- `insert_child`
- `delete_child`
- `accepts_objects`

In Constraint widget classes:

- `constraint_size`

In Shell widget classes:

- `root_geometry_manager`

With downward superclass chaining, the invocation of an operation first accesses the field from the Object, RectObj, and Core class structures, then from the subclass structure, and so on down the class chain to that widget’s class structure. These superclass-to-subclass fields are:

- `class_part_initialize`
- `get_values_hook`
- `initialize`
- `initialize_hook`
- `set_values`
- `set_values_hook`
- `resources`
In addition, for subclasses of Constraint, the following fields of the ConstraintClassPart and ConstraintClassExtensionRec structures are chained from the Constraint class down to the subclass:

- resources
- initialize
- set_values
- get_values_hook

With upward superclass chaining, the invocation of an operation first accesses the field from the widget class structure, then from the superclass structure, and so on up the class chain to the Core, RectObj, and Object class structures. The subclass-to-superclass fields are

- destroy
- actions

For subclasses of Constraint, the following field of ConstraintClassPart is chained from the subclass up to the Constraint class:

- destroy

1.6.8. Class Initialization: class_initialize and class_part_initialize Procedures

Many class records can be initialized completely at compile or link time. In some cases, however, a class may need to register type converters or perform other sorts of once-only runtime initialization.

Because the C language does not have initialization procedures that are invoked automatically when a program starts up, a widget class can declare a class_initialize procedure that will be automatically called exactly once by the Intrinsics. A class initialization procedure pointer is of type XtProc:

```c
typedef void (*XtProc)(void);
```

A widget class indicates that it has no class initialization procedure by specifying NULL in the class_initialize field.

In addition to the class initialization that is done exactly once, some classes perform initialization for fields in their parts of the class record. These are performed not just for the particular class but for subclasses as well, and are done in the class's class part initialization procedure, a pointer to which is stored in the class_part_initialize field. The class_part_initialize procedure pointer is of type XtWidgetClassProc:

```c
typedef void (*XtWidgetClassProc)(WidgetClass);
```

`widget_class` Points to the class structure for the class being initialized.

During class initialization, the class part initialization procedures for the class and all its superclasses are called in superclass-to-subclass order on the class record. These procedures have the responsibility of doing any dynamic initializations necessary to their class's part of the record. The most common is the resolution of any inherited methods defined in the class. For example, if a widget class C has superclasses Core, Composite, A, and B, the class record for C first is passed to Core's class_part_initialize procedure. This resolves any inherited Core methods and compiles the textual representations of the resource list and action table that are defined in the class record. Next, Composite's class_part_initialize procedure is called to initialize the composite part of C's class record. Finally, the class_part_initialize procedures for A, B, and C, in that order, are called. For further information, see Section 1.6.9. Classes that do not define any new class fields or that need no extra processing for them can specify NULL
in the class_part_initialize field.

All widget classes, whether they have a class initialization procedure or not, must start with their class_initialized field False.

The first time a widget of a class is created, XtCreateWidget ensures that the widget class and all superclasses are initialized, in superclass-to-subclass order, by checking each class_initialized field and, if it is False, by calling the class_initialize and the class_part_initialize procedures for the class and all its superclasses. The Intrinsics then set the class_initialized field to a nonzero value. After the one-time initialization, a class structure is constant.

The following example provides the class initialization procedure for a Label class.

```c
static void ClassInitialize()
{
    XtSetTypeConverter(XtRString, XtRJustify, CvtStringToJustify,
        NULL, 0, XtCacheNone, NULL);
}
```

1.6.9. Initializing a Widget Class

A class is initialized when the first widget of that class or any subclass is created. To initialize a widget class without creating any widgets, use XtInitializeWidgetClass.

```c
void XtInitializeWidgetClass(object_class)

    WidgetClass object_class;
```

object_class Specifies the object class to initialize. May be objectClass or any subclass thereof.

If the specified widget class is already initialized, XtInitializeWidgetClass returns immediately.

If the class initialization procedure registers type converters, these type converters are not available until the first object of the class or subclass is created or XtInitializeWidgetClass is called (see Section 9.6).

1.6.10. Inheritance of Superclass Operations

A widget class is free to use any of its superclass’s self-contained operations rather than implementing its own code. The most frequently inherited operations are expose realize insert_child delete_child geometry_manager set_values_almost

To inherit an operation _xyz_, specify the constant XtInheritXyz in your class record.

Every class that declares a new procedure in its widget class part must provide for inheriting the procedure in its class_part_initialize procedure. The chained operations declared in Core and Constraint records are never inherited. Widget classes that do nothing beyond what their superclass does specify NULL for chained procedures in their class records.

Inheriting works by comparing the value of the field with a known, special value and by copying in the superclass’s value for that field if a match occurs. This special value, called the inheritance constant, is usually the Intrinsics internal value _XtInherit cast to the appropriate type. _XtInherit is a procedure that issues an error message if it is actually called.
For example, CompositeP.h contains these definitions:

```c
#define XtlnheritGeometryManager ((XtGeometryHandler) _XtInherit)
#define XtlnheritChangeManaged ((XtWidgetProc) _XtInherit)
#define XtlnheritInsertChild ((XtArgsProc) _XtInherit)
#define XtlnheritDeleteChild ((XtWidgetProc) _XtInherit)
```

Composite’s class_part_initialize procedure begins as follows:

```c
static void CompositeClassPartInitialize(widgetClass)
{
    WidgetGass widgetClass;
    CompositeWidgetClass wc = (CompositeWidgetClass)widgetClass;
    CompositeWidgetClass super = (CompositeWidgetClass)wc->core_class.superclass;

    if (wc->composite_class.geometry_manager == XtlnheritGeometryManager) {
        wc->composite_class.geometry_manager = super->composite_class.geometry_manager;
    }

    if (wc->composite_class.change_managed == XtlnheritChangeManaged) {
        wc->composite_class.change_managed = super->composite_class.change_managed;
    }

    ...
    ...
```

Nonprocedure fields may be inherited in the same manner as procedure fields. The class may declare any reserved value it wishes for the inheritance constant for its new fields. The following inheritance constants are defined:

For Core:
- XtInheritRealize
- XtInheritResize
- XtInheritExpose
- XtInheritSetValueAlmost
- XtInheritAcceptFocus
- XtInheritQueryGeometry
- XtInheritTranslations
- XtInheritDisplayAccelerator

For Composite:
- XtInheritGeometryManager
- XtInheritChangeManaged
- XtInheritInsertChild
- XtInheritDeleteChild

For Shell:
- XtInheritRootGeometryManager

1.6.11. Invocation of Superclass Operations

A widget sometimes needs to call a superclass operation that is not chained. For example, a widget’s expose procedure might call its superclass’s `expose` and then perform a little more
work on its own. For example, a Composite class with predefined managed children can implement `insert_child` by first calling its superclass’s `insert_child` and then calling `XtManageChild` to add the child to the managed set.

**Note**

A class method should not use `XtSuperclass` but should instead call the class method of its own specific superclass directly through the superclass record. That is, it should use its own class pointers only, not the widget’s class pointers, as the widget’s class may be a subclass of the class whose implementation is being referenced.

This technique is referred to as *enveloping* the superclass’s operation.

### 1.6.12. Class Extension Records

It may be necessary at times to add new fields to already existing widget class structures. To permit this to be done without requiring recompilation of all subclasses, the last field in a class part structure should be an extension pointer. If no extension fields for a class have yet been defined, subclasses should initialize the value of the extension pointer to NULL.

If extension fields exist, as is the case with the Composite, Constraint and Shell classes, subclasses can provide values for these fields by setting the `extension` pointer for the appropriate part in their class structure to point to a statically declared extension record containing the additional fields. Setting the `extension` field is never mandatory; code that uses fields in the extension record must always check the `extension` field and take some appropriate default action if it is NULL.

In order to permit multiple subclasses and libraries to chain extension records from a single `extension` field, extension records should be declared as a linked list and each extension record definition should contain the following four fields at the beginning of the structure declaration:

```c
struct {
    XtPointer next_extension;
    XrmQuark record_type;
    long version;
    Cardinal record_size;
};
```

- `next_extension` Specifies the next record in the list, or NULL.
- `record_type` Specifies the particular structure declaration to which each extension record instance conforms.
- `version` Specifies a version id symbolic constant supplied by the definer of the structure.
- `record_size` Specifies the total number of bytes allocated for the extension record.

The `record_type` field identifies the contents of the extension record and is used by the definer of the record to locate its particular extension record in the list. The `record_type` field is normally assigned the result of `XrmStringToQuark` for a registered string constant. The Intrinsics reserve all record type strings beginning with the two characters “XT” for future standard uses. The value `NULLQUARK` may also be used by the class part owner in extension records attached to its own class part extension field to identify the extension record unique to that particular class.

The `version` field is an owner-defined constant that may be used to identify binary files that have been compiled with alternate definitions of the remainder of the extension record data structure. The private header file for a widget class should provide a symbolic constant for subclasses to use to initialize this field. The `record_size` field value includes the four common
header fields and should normally be initialized with sizeof().
Any value stored in the class part extension fields of CompositeClassPart, ConstraintClassPart, or ShellClassPart must point to an extension record conforming to this definition.
Chapter 2

Widget Instantiation

A hierarchy of widget instances constitutes a widget tree. The shell widget returned by XtAppCreateShell is the root of the widget tree instance. The widgets with one or more children are the intermediate nodes of that tree, and the widgets with no children of any kind are the leaves of the widget tree. With the exception of pop-up children (see Chapter 5), this widget tree instance defines the associated X Window tree.

Widgets can be either composite or primitive. Both kinds of widgets can contain children, but the Intrinsics provide a set of management mechanisms for constructing and interfacing between composite widgets, their children, and other clients.

Composite widgets, that is, members of the class compositeWidgetClass, are containers for an arbitrary but widget implementation-defined collection of children, which may be instantiated by the composite widget itself, by other clients, or by a combination of the two. Composite widgets also contain methods for managing the geometry (layout) of any child widget. Under unusual circumstances, a composite widget may have zero children, but it usually has at least one. By contrast, primitive widgets that contain children typically instantiate specific children of known classes themselves and do not expect external clients to do so. Primitive widgets also do not have general geometry management methods.

In addition, the Intrinsics recursively perform many operations (for example, realization and destruction) on composite widgets and all their children. Primitive widgets that have children must be prepared to perform the recursive operations themselves on behalf of their children.

A widget tree is manipulated by several Intrinsics functions. For example, XtRealizeWidget traverses the tree downward and recursively realizes all pop-up widgets and children of composite widgets. XtDestroyWidget traverses the tree downward and destroys all pop-up widgets and children of composite widgets. The functions that fetch and modify resources traverse the tree upward and determine the inheritance of resources from a widget's ancestors. XtMakeGeometryRequest traverses the tree up one level and calls the geometry manager that is responsible for a widget child's geometry.

To facilitate upward traversal of the widget tree, each widget has a pointer to its parent widget. The Shell widget that XtAppCreateShell returns has a parent pointer of NULL.

To facilitate downward traversal of the widget tree, the children field of each composite widget is a pointer to an array of child widgets, which includes all normal children created, not just the subset of children that are managed by the composite widget's geometry manager. Primitive widgets that instantiate children are entirely responsible for all operations that require downward traversal below themselves. In addition, every widget has a pointer to an array of pop-up children.

2.1. Initializing the X Toolkit

Before an application can call any Intrinsics function other than XtSetLanguageProc, it must initialize the Intrinsics by using

- XtToolkitInitialize, which initializes the Intrinsics internals.
- XtCreateApplicationContext, which initializes the per-application state.
- XtDisplayInitialize or XtOpenDisplay, which initializes the per-display state.
- XtAppCreateShell, which creates the root of a widget tree.
or an application can call the convenience procedure \texttt{XtAppInitialize} which combines the functions of the preceding procedures. An application wishing to use the ANSI C locale mechanism should call \texttt{XtSetLanguageProc} prior to calling \texttt{XtDisplayInitialize}, \texttt{XtOpenDisplay}, or \texttt{XtAppInitialize}.

Multiple instances of X Toolkit applications may be implemented in a single address space. Each instance needs to be able to read input and dispatch events independently of any other instance. Further, an application instance may need multiple display connections to have widgets on multiple displays. From the application's point of view, multiple display connections usually are treated together as a single unit for purposes of event dispatching. To accommodate both requirements, the Intrinsics define application contexts, each of which provides the information needed to distinguish one application instance from another. The major component of an application context is a list of one or more \texttt{X Display} pointers for that application. The Intrinsics handle all display connections within a single application context simultaneously, handling input in a round-robin fashion. The application context type \texttt{XtAppContext} is opaque to clients.

To initialize the Intrinsics internals, use \texttt{XtToolkitInitialize}.

\begin{verbatim}
void XtToolkitInitialize()
\end{verbatim}

The semantics of calling \texttt{XtToolkitInitialize} more than once are undefined.

To create an application context, use \texttt{XtCreateApplicationContext}.

\begin{verbatim}
XtAppContext XtCreateApplicationContext()
\end{verbatim}

The \texttt{XtCreateApplicationContext} function returns an application context, which is an opaque type. Every application must have at least one application context.

To destroy an application context and close any remaining display connections in it, use \texttt{XtDestroyApplicationContext}.

\begin{verbatim}
void XtDestroyApplicationContext(app_context)
    XtAppContext app_context;
\end{verbatim}

\texttt{app_context} specifies the application context.

The \texttt{XtDestroyApplicationContext} function destroys the specified application context as soon as it is safe to do so. If called from within an event dispatch (for example, in a callback procedure), \texttt{XtDestroyApplicationContext} does not destroy the application context until the dispatch is complete.

To get the application context in which a given widget was created, use \texttt{XtWidgetToApplicationContext}.

\begin{verbatim}
XtAppContext XtWidgetToApplicationContext(w)
    Widget w;
\end{verbatim}

\texttt{w} specifies the widget for which you want the application context. Must be of class \texttt{Object} or any subclass thereof.

The \texttt{XtWidgetToApplicationContext} function returns the application context for the specified widget.

To initialize a display and add it to an application context, use \texttt{XtDisplayInitialize}.
void XtDisplayInitialize(app_context, display, application_name, application_class,
    options, num_options, argc, argv)

XtAppContext app_context;
Display *display;
String application_name;
String application_class;
XrmOptionDescRec *options;
Cardinal num_options;
int *argc;
String *argv;

app_context  Specifies the application context.
display      Specifies a previously opened display connection. Note that a single
display connection can be in at most one application context.
application_name Specifies the name of the application instance.
application_class Specifies the class name of this application, which is usually the generic
                     name for all instances of this application.
options      Specifies how to parse the command line for any application-specific
                     resources. The options argument is passed as a parameter to XrmParseCommand. For further information, see Section 15.9 in Xlib – C
                     Language X Interface and Section 2.4 of this specification.
num_options  Specifies the number of entries in the options list.
argc         Specifies a pointer to the number of command line parameters.
argv          Specifies the list of command line parameters.

The XtDisplayInitialize function retrieves the language string to be used for the specified
display (see Section 11.11), calls the language procedure (if set) with that language string,
built the resource database for the default screen, calls the Xlib XrmParseCommand function
to parse the command line, and performs other per-display initialization. After XrmParseCommand
has been called, argc and argv contain only those parameters that were not in the
standard option table or in the table specified by the options argument. If the modified argc is
not zero, most applications simply print out the modified argv along with a message listing the
allowable options. On POSIX-based systems, the application name is usually the final com-
ponent of argv[0]. If the synchronous resource is True, XtDisplayInitialize calls the Xlib
XSynchronize function to put Xlib into synchronous mode for this display connection and any
others currently open in the application context. See Sections 2.3 and 2.4 for details on the
application_name, application_class, options, and num_options arguments.

XtDisplayInitialize calls XrmSetDatabase to associate the resource database of the default
screen with the display before returning.
To open a display, initialize it and then add it to an application context, use XtOpenDisplay.

```
Display *XtOpenDisplay(app_context, display_string, application_name, application_class, options, argc, argv) {
    XtAppContext app_context;
    String display_string;
    String application_name;
    String application_class;
    XrmOptionDescRec *options;
    Cardinal num_options;
    int *argc;
    String *argv;

    app_context Specifies the application context.
    display_string Specifies the display string, or NULL.
    application_name Specifies the name of the application instance, or NULL.
    application_class Specifies the class name of this application, which is usually the generic name for all instances of this application.
    options Specifies how to parse the command line for any application-specific resources. The option argument is passed as a parameter to XrmParseCommand.
    num_options Specifies the number of entries in the options list.
    argc Specifies a pointer to the number of command line parameters.
    argv Specifies the list of command line parameters.
```

The XtOpenDisplay function calls XOpenDisplay with the specified display_string. If display_string is NULL, XtOpenDisplay uses the current value of the -display option specified in argv. If no display is specified in argv, the user's default display is retrieved from the environment. On POSIX-based systems, this is the value of the DISPLAY environment variable.

If this succeeds, XtOpenDisplay then calls XtDisplayInitialize and passes it the opened display and the value of the -name option specified in argv as the application name. If no -name option is specified and application_name is non-NULL, application_name is passed to XtDisplayInitialize. If application_name is NULL and if the environment variable RESOURCE_NAME is set, the value of RESOURCE_NAME is used. Otherwise, the application name is the name used to invoke the program. On implementations that conform to ANSI C Hosted Environment support, the application name will be argv[0] less any directory and file type components, that is, the final component of argv[0], if specified. If argv[0] does not exist or is the empty string, the application name is "main". XtOpenDisplay returns the newly opened display or NULL if it failed.

To close a display and remove it from an application context, use XtCloseDisplay.

```
void XtCloseDisplay(display) {
    Display *display;

    display Specifies the display.
```

The XtCloseDisplay function calls XCloseDisplay with the specified display as soon as it is safe to do so. If called from within an event dispatch (for example, a callback procedure), XtCloseDisplay does not close the display until the dispatch is complete. Note that applications need only call XtCloseDisplay if they are to continue executing after closing the display; otherwise, they should call XtDestroyApplicationContext or just exit.
2.2. Establishing the Locale

Resource databases are specified to be created in the current process locale. During display initialization prior to creating the per-screen resource database, the Intrinsics will call out to a specified application procedure to set the locale according to options found on the command line or in the per-display resource specifications.

The callout procedure provided by the application is of type XtLanguageProc.

typedef String (*XtLanguageProc)(Display*, String, XtPointer);

Display *display;
String language;
XtPointer client_data;

display Passes the display.
language Passes the initial language value obtained from the command line or server per-display resource specifications.
client_data Passes the additional client data specified in the call to XtSetLanguageProc.

The language procedure allows an application to set the locale to the value of the language resource determined by XtDisplayInitialize. The function returns a new language string that will be subsequently used by XtDisplayInitialize to establish the path for loading resource files. The returned string will be copied by the Intrinsics into new memory.

Initially, no language procedure is set by the Intrinsics. To set the language procedure for use by XtDisplayInitialize use XtSetLanguageProc.

XtLanguageProc XtSetLanguageProc(app_context, proc, client_data)

XtAppContext app_context;
XtLanguageProc proc;
XtPointer client_data;

app_context Specifies the application context in which the language procedure is to be used, or NULL.
proc Specifies the language procedure.
client_data Specified additional client data to be passed to the language procedure when it is called.

XtSetLanguageProc sets the language procedure that will be called from XtDisplayInitialize for all subsequent Displays initialized in the specified application context. If app_context is NULL, the specified language procedure is registered in all application contexts created by the calling process, including any future application contexts that may be created. If proc is NULL a default language procedure is registered. XtSetLanguageProc returns the previously registered language procedure. If a language procedure has not yet been registered, the return value is unspecified but if this return value is used in a subsequent call to XtSetLanguageProc, it will cause the default language procedure to be registered.

The default language procedure does the following:

- Sets the locale according to the environment. On ANSI C-based systems this is done by calling setlocale( LC_ALL, language ). If an error is encountered a warning message is issued with XtWarning.
- Calls XSupportsLocale to verify that the current locale is supported. If the locale is not supported, a warning message is issued with XtWarning and the locale is set to "C".
- Calls XSetLocaleModifiers specifying the empty string.
• Returns the value of the current locale. On ANSI C-based systems this is the return value from a final call to setlocale( LC_ALL, NULL ).

A client wishing to use this mechanism to establish locale can do so by calling XtSetLanguageProc prior to XtDisplayInitialize, as in the following example.

Widget top;
XtSetLanguageProc(NULL, NULL, NULL);
top = XtAppInitialize( ... );
...

2.3. Loading the Resource Database

The XtDisplayInitialize function first determines the language string to be used for the specified display. It then creates a resource database for the default screen of the display by combining the following sources in order, with the entries in the first named source having highest precedence:

• Application command line (argc, argv).
• Per-host user environment resource file on the local host.
• Per-screen resource specifications from the server.
• Per-display resource specifications from the server or from the user preference file on the local host.
• Application-specific user resource file on the local host.
• Application-specific class resource file on the local host.

When the resource database for a particular screen on the display is needed (either internally, or when XtScreenDatabase is called), it is created in the following manner using the sources listed above in the same order:

• A temporary database, the "server resource database", is created from the string returned by XResourceManagerString or, if XResourceManagerString returns NULL, the contents of a resource file in the user’s home directory. On POSIX-based systems, the usual name for this user preference resource file is $HOME/.Xdefaults.

• If a language procedure has been set, XtDisplayInitialize first searches the command line for the option "-xnlLanguage", or for a -xnm option that specifies the xnlLanguage/XnlLanguage resource, as specified by Section 2.4. If such a resource is found, the value is assumed to be entirely in XPCS, the X Portable Character Set. If neither option is specified on the command line, XtDisplayInitialize queries the server resource database (which is assumed to be entirely in XPCS) for the resource nowf.xnlLanguage.class C&w.Xn!Language where name and Class are the application_name and application_class specified to XtDisplayInitialize. The language procedure is then invoked with the resource value if found, else the empty string. The string returned from the language procedure is saved for all future references in the Intrinsics that require the per-display language string.

• The screen resource database is initialized by parsing the command line in the manner specified by Section 2.4.
If a language procedure has not been set, the initial database is then queried for the resource \texttt{name.xnlLanguage}, class \texttt{Class.XnlLanguage} as specified above. If this database query fails, the server resource database is queried; if this query also fails, the language is determined from the environment; on POSIX-based systems, this is done by retrieving the value of the \texttt{LANG} environment variable. If no language string is found, the empty string is used. This language string is saved for all future references in the Intrinsics that require the per-display language string.

After determining the language string, the user's environment resource file is then merged into the initial resource database if the file exists. This file is user-, host-, and process-specific and is expected to contain user preferences that are to override those specifications in the per-display and per-screen resources. On POSIX-based systems, the user's environment resource file name is specified by the value of the \texttt{XENVIRONMENT} environment variable. If this environment variable does not exist, the user's home directory is searched for a file named \texttt{.Xdefaults-host}, where \texttt{host} is the host name of the machine on which the application is running.

The per-screen resource specifications are then merged into the screen resource database, if they exist. These specifications are the string returned by \texttt{XScreenResourceString} for the respective screen and are owned entirely by the user.

Next, the server resource database created earlier is merged into the screen resource database. The server property, and corresponding user preference file, are owned and constructed entirely by the user.

The application-specific user resource file from the local host is then merged into the screen resource database. This file contains user customizations and is stored in a directory owned by the user. Either the user or the application or both can store resource specifications in the file. Each should be prepared to find and respect entries made by the other. The file name is found by calling \texttt{XrmSetDatabase} with the current screen resource database, after preserving the original display-associated database, then calling \texttt{XtResolvePathname} with the parameters (\texttt{display}, \texttt{NULL}, \texttt{NULL}, \texttt{NULL}, \texttt{path}, \texttt{NULL}, 0, \texttt{NULL}) where \texttt{path} is defined in an operating-system-specific way. On POSIX-based systems, \texttt{path} is defined to be the value of the environment variable \texttt{XUSERFILESEAR-CHPATH} if this is defined. If \texttt{XUSERFILESEAR-CHPATH} is not defined, an implementation-dependent default value is used. This default value is constrained in the following manner:

If the environment variable \texttt{XAPPLRESDIR} is not defined, the default \texttt{XUSERFILESEAR-CHPATH} must contain at least six entries. These entries must contain \texttt{$HOME} as the directory prefix, plus the following substitutions:

\begin{enumerate}
\item \texttt{\%C, \%N, \%L} or \texttt{\%C, \%N, \%l, \%t, \%c}
\item \texttt{\%C, \%N, \%l}
\item \texttt{\%C, \%N}
\item \texttt{\%N, \%L} or \texttt{\%N, \%l, \%t, \%c}
\item \texttt{\%N, \%l}
\item \texttt{\%N}
\end{enumerate}

The order of these six entries within the path must be as given above. The order and use of substitutions within a given entry is implementation dependent.
- If $XAPPLRESDIR$ is defined, the default $XUSERFILESEARCHPATH$ must contain at least seven entries. These entries must contain the following directory prefixes and substitutions:

1. $XAPPLRESDIR$ with $%C$, $%N$, $%L$ or $%C$, $%N$, $%l$, $%t$, $%c$
2. $XAPPLRESDIR$ with $%C$, $%N$, $%l$
3. $XAPPLRESDIR$ with $%C$, $%N$
4. $XAPPLRESDIR$ with $%N$, $%L$ or $%N$, $%l$, $%t$, $%c$
5. $XAPPLRESDIR$ with $%N$, $%l$
6. $XAPPLRESDIR$ with $%N$
7. $SHOME$ with $%N$

The order of these seven entries within the path must be as given above. The order and use of substitutions within a given entry is implementation dependent.

• Lastly, the application-specific class resource file from the local host is merged into the screen resource database. This file is owned by the application and is usually installed in a system directory when the application is installed. It may contain site-wide customizations specified by the system manager. The name of the application class resource file is found by calling $XtResolvePathname$ with the parameters ($display$, "app-defaults", NULL, NULL, NULL, NULL, 0, NULL). This file is expected to be provided by the developer of the application and may be required for the application to function properly. A simple application that wants to be assured of having a minimal set of resources in the absence of its class resource file can declare fallback resource specifications with $XtAppSetFallbackResources$. Note that the customization substitution string is retrieved dynamically by $XtResolvePathname$ so that the resolved file name of the application class resource file can be affected by any of the earlier sources for the screen resource database, even though the contents of the class resource file have lowest precedence. After calling $XtResolvePathname$, the original display-associated database is restored.

To obtain the resource database for a particular screen, use $XtScreenDatabase$.

$XrmDatabase XtScreenDatabase(screen)$

Screen *screen;

$screen$ Specifies the screen whose resource database is to be returned.

The $XtScreenDatabase$ function returns the fully merged resource database as specified above, associated with the specified screen. If the specified $screen$ does not belong to a $Display$ initialized by $XtDisplayInitialize$, the results are undefined.

To obtain the default resource database associated with a particular display, use $XtDatabase$.

$XrmDatabase XtDatabase(display)$

Display *display;

display Specifies the display.

The $XtDatabase$ function is equivalent to $XrmGetDatabase$. It returns the database associated with the specified display, or NULL if a database has not been set.

To specify a default set of resource values that will be used to initialize the resource database if no application-specific class resource file is found (the last of the six sources listed above),
use XtAppSetFallbackResources.

void XtAppSetFallbackResources(app_context, specification_list)
    XtAppContext app_context;
    String *specification_list;

app_context  Specifies the application context in which the fallback specifications will be used.

specification_list  Specifies a NULL-terminated list of resource specifications to preload the database, or NULL.

Each entry in specification_list points to a string in the format of XrmPutLineResource. Following a call to XtAppSetFallbackResources, when a resource database is being created for a particular screen and the Intrinsics are not able to find or read an application-specific class resource file according to the rules given above and if specification_list is not NULL the resource specifications in specification_list will be merged into the screen resource database in place of the application-specific class resource file. XtAppSetFallbackResources is not required to copy specification_list; the caller must ensure that the contents of the list and of the strings addressed by the list remain valid until all displays are initialized or until XtAppSetFallbackResources is called again. The value NULL for specification_list removes any previous fallback resource specification for the application context. The intended use for fallback resources is to provide a minimal number of resources that will make the application usable (or at least terminate with helpful diagnostic messages) when some problem exists in finding and loading the application defaults file.

2.4. Parsing the Command Line

The XtOpenDisplay function first parses the command line for the following options:

--display  Specifies the display name for XOpenDisplay.

--name  Sets the resource name prefix, which overrides the application name passed to XtOpenDisplay.

--xnllanguage  Specifies the initial language string for establishing locale and for finding application class resource files.

XtDisplayInitialize has a table of standard command line options that are passed to XrmParseCommand for adding resources to the resource database, and it takes as a parameter additional application-specific resource abbreviations. The format of this table is described in Section 15.9 in Xlib – C Language X Interface.

typedef enum {
    XrmoptionNoArg,  /* Value is specified in OptionDescRec.value */
    XrmoptionIsArg,  /* Value is the option string itself */
    XrmoptionStickyArg,  /* Value is characters immediately following option */
    XrmoptionSepArg,  /* Value is next argument in argv */
    XrmoptionResArg,  /* Use the next argument as input to XrmPutLineResource*/
    XrmoptionSkipArg,  /* Ignore this option and the next argument in argv */
    XrmoptionSkipNArgs,  /* Ignore this option and the next */
    XrmoptionSkipLine,  /* Ignore this option and the rest of argv */
} XrmOptionKind;

typedef struct {
    char *option;  /* Option name in argv */
    char *specifier,  /* Resource name (without application name) */
    XrmOptionKind argKind;  /* Location of the resource value */
    XPointer value;  /* Value to provide if XrmoptionNoArg */
} XrmDescRec;
The standard table contains the following entries:

<table>
<thead>
<tr>
<th>Option String</th>
<th>Resource Name</th>
<th>Argument Kind</th>
<th>Resource Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-background</td>
<td>*background</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-bd</td>
<td>*borderColor</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-bg</td>
<td>*background</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-borderwidth</td>
<td>.borderWidth</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-bordercolor</td>
<td>*borderColor</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-bw</td>
<td>.borderWidth</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-display</td>
<td>.display</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-fg</td>
<td>*foreground</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-fn</td>
<td>*font</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-font</td>
<td>*font</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-foreground</td>
<td>*foreground</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-geometry</td>
<td>.geometry</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-iconic</td>
<td>.iconic</td>
<td>NoArg</td>
<td>&quot;true&quot;</td>
</tr>
<tr>
<td>-name</td>
<td>.name</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-reverse</td>
<td>.reverseVideo</td>
<td>NoArg</td>
<td>&quot;on&quot;</td>
</tr>
<tr>
<td>-rv</td>
<td>.reverseVideo</td>
<td>NoArg</td>
<td>&quot;on&quot;</td>
</tr>
<tr>
<td>+rv</td>
<td>.reverseVideo</td>
<td>NoArg</td>
<td>&quot;off&quot;</td>
</tr>
<tr>
<td>-selectionTimeout</td>
<td>selectionTimeout</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-synchronous</td>
<td>.synchronous</td>
<td>NoArg</td>
<td>&quot;on&quot;</td>
</tr>
<tr>
<td>+synchronous</td>
<td>.synchronous</td>
<td>NoArg</td>
<td>&quot;off&quot;</td>
</tr>
<tr>
<td>-title</td>
<td>.title</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-xnlanguage</td>
<td>.xnlLanguage</td>
<td>SepArg</td>
<td>next argument</td>
</tr>
<tr>
<td>-xrm</td>
<td>next argument</td>
<td>ResArg</td>
<td>next argument</td>
</tr>
</tbody>
</table>

Note that any unique abbreviation for an option name in the standard table or in the application table is accepted.

If reverseVideo is True, the values of XtDefaultForeground and XtDefaultBackground are exchanged for all screens on the Display.

The value of the synchronous resource specifies whether or not Xlib is put into synchronous mode. If a value is found in the resource database during display initialization, XtDisplayInitialize makes a call to XSynchronize for all display connections currently open in the application context. Therefore, when multiple displays are initialized in the same application context, the most recent value specified for the synchronous resource is used for all displays in the application context.

The value of the selectionTimeout resource applies to all displays opened in the same application context. When multiple displays are initialized in the same application context, the most recent value specified is used for all displays in the application context.

The -xrm option provides a method of setting any resource in an application. The next argument should be a quoted string identical in format to a line in the user resource file. For example, to give a red background to all command buttons in an application named xmh, you can start it up as

```
xmh -xrm 'xmh*Command.background: red'
```

When it parses the command line, XtDisplayInitialize merges the application option table with the standard option table before calling the Xlib XrmParseCommand function. An entry in
the application table with the same name as an entry in the standard table overrides the standard table entry. If an option name is a prefix of another option name, both names are kept in the merged table. The Intrinsics reserve all option names beginning with the characters "-xt" for future standard uses.

2.5. Creating Widgets

The creation of widget instances is a three-phase process:

1. The widgets are allocated and initialized with resources and are optionally added to the managed subset of their parent.
2. All composite widgets are notified of their managed children in a bottom-up traversal of the widget tree.
3. The widgets create X windows, which then are mapped.

To start the first phase, the application calls XtCreateWidget for all its widgets and adds some (usually, most or all) of its widgets to their respective parents' managed set by calling XtManageChild. To avoid an $O(n^2)$ creation process where each composite widget lays itself out each time a widget is created and managed, parent widgets are not notified of changes in their managed set during this phase.

After all widgets have been created, the application calls XtRealizeWidget with the top-level widget to execute the second and third phases. XtRealizeWidget first recursively traverses the widget tree in a postorder (bottom-up) traversal and then notifies each composite widget with one or more managed children by means of its change_managed procedure.

Notifying a parent about its managed set involves geometry layout and possibly geometry negotiation. A parent deals with constraints on its size imposed from above (for example, when a user specifies the application window size) and suggestions made from below (for example, when a primitive child computes its preferred size). One difference between the two can cause geometry changes to ripple in both directions through the widget tree. The parent may force some of its children to change size and position and may issue geometry requests to its own parent in order to better accommodate all its children. You cannot predict where anything will go on the screen until this process finishes.

Consequently, in the first and second phases, no X windows are actually created, because it is likely that they will get moved around after creation. This avoids unnecessary requests to the X server.

Finally, XtRealizeWidget starts the third phase by making a preorder (top-down) traversal of the widget tree, allocates an X window to each widget by means of its realize procedure, and finally maps the widgets that are managed.

2.5.1. Creating and Merging Argument Lists

Many Intrinsics functions may be passed pairs of resource names and values. These are passed as an arglist, a pointer to an array of Arg structures, which contains

```c
typedef struct {
    String name;
    XtArgVal value;
} Arg, *ArgList;
```

where XtArgVal is as defined in Section 1.5.

If the size of the resource is less than or equal to the size of an XtArgVal, the resource value is stored directly in value; otherwise, a pointer to it is stored in value.

To set values in an ArgList, use XtSetArg.
void XtSetArg(arg, name, value)
  Arg arg;
  String name;
  XtArgVal value;

arg      Specifies the name/value pair to set.
name     Specifies the name of the resource.
value    Specifies the value of the resource if it will fit in an XtArgVal, else the
          address.

The XtSetArg function is usually used in a highly stylized manner to minimize the probability
of making a mistake; for example:

Arg args[20];
int n;

n = 0;
XtSetArg(args[n], XtNheight, 100); n++;
XtSetArg(args[n], XtNwidth, 200); n++;
XtSetValues(widget, args, n);

Alternatively, an application can statically declare the argument list and use XtNumber:

static Args args[] = {
  {XtNheight, (XtArgVal) 100},
  {XtNwidth, (XtArgVal) 200},
};
XtSetValues(Widget, args, XtNumber(args));

Note that you should not use expressions with side effects such as auto-increment or auto-
decrement within the first argument to XtSetArg. XtSetArg can be implemented as a macro
that evaluates the first argument twice.

To merge two arglist arrays, use XtMergeArgLists.

ArgList XtMergeArgLists(args1, num_args1, args2, num_args2)
  ArgList args1;
  Cardinal num_args1;
  ArgList args2;
  Cardinal num_args2;

args1     Specifies the first argument list.
num_args1 Specifies the number of entries in the first argument list.
args2     Specifies the second argument list.
num_args2 Specifies the number of entries in the second argument list.

The XtMergeArgLists function allocates enough storage to hold the combined arglist arrays
and copies them into it. Note that it does not check for duplicate entries. The length of the
returned list is the sum of the lengths of the specified lists. When it is no longer needed, free
the returned storage by using XtFree.

All Intrinsics interfaces that require ArgList arguments have analogs conforming to the ANSI
C variable argument list (traditionally called “varargs”) calling convention. The name of the
analog is formed by prefixing “Va” to the name of the corresponding ArgList procedure;
e.g., XtVaCreateWidget. Each procedure named XtVa*something takes as its last arguments,
in place of the corresponding ArgList/ Cardinal parameters, a variable parameter list of
resource name and value pairs where each name is of type String and each value is of type XtArgVal. The end of the list is identified by a name entry containing NULL. Developers writing in the C language wishing to pass resource name and value pairs to any of these interfaces may use the ArgList and varargs forms interchangeably.

Two special names are defined for use only in varargs lists: XtVaTypedArg and XtVaNestedList.

#define XtVaTypedArg "XtVaTypedArg"

If the name XtVaTypedArg is specified in place of a resource name, then the following four arguments are interpreted as a name/type/value/size tuple where name is of type String, type is of type String, value is of type XtArgVal, and size is of type int. When a varargs list containing XtVaTypedArg is processed, a resource type conversion (see Section 9.6) is performed if necessary to convert the value into the format required by the associated resource. If type is XtRString then value contains a pointer to the string and size contains the number of bytes allocated, including the trailing null byte. If type is not XtRString, then if size is less than or equal to sizeof(XtArgVal), the value should be the data cast to the type XtArgVal, otherwise value is a pointer to the data. If the type conversion fails for any reason, a warning message is issued and the list entry is skipped.

#define XtVaNestedList "XtVaNestedList"

If the name XtVaNestedList is specified in place of a resource name, then the following argument is interpreted as an XtVarArgsList value, which specifies another varargs list that is logically inserted into the original list at the point of declaration. The end of the nested list is identified with a name entry containing NULL. Varargs lists may nest to any depth.

To dynamically allocate a varargs list for use with XtVaNestedList in multiple calls, use XtVaCreateArgsList.

typedef XtPointer XtVarArgsList;

XtVarArgsList XtVaCreateArgsList(unused, ...)

XtPointer unused;

unused This argument is not currently used and must be specified as NULL.
...

Specifies a variable parameter list of resource name and value pairs.

The XtVaCreateArgsList function allocates memory and copies its arguments into a single list pointer, which may be used with XtVaNestedList. The end of both lists is identified by a name entry containing NULL. Any entries of type XtVaTypedArg are copied as specified without applying conversions. Data passed by reference (including Strings) are not copied, only the pointers themselves; the caller must ensure that the data remain valid for the lifetime of the created varargs list. The list should be freed using XtFree when no longer needed.

Use of resource files and the resource database is generally encouraged over lengthy arglist or varargs lists whenever possible in order to permit modification without recompilation.

2.5.2. Creating a Widget Instance

To create an instance of a widget, use XtCreateWidget.
Widget XtCreateWidget(name, object_class, parent, args, num_args)
    String name;
    WidgetClass object_class;
    Widget parent;
    ArgList args;
    Cardinal num_args;

name Specifies the resource instance name for the created widget, which is used for
retrieving resources and, for that reason, should not be the same as any other
widget that is a child of the same parent.

object_class Specifies the widget class pointer for the created object. Must be objectClass
or any subclass thereof.

parent Specifies the parent widget. Must be of class Object or any subclass thereof.

args Specifies the argument list to override any other resource specifications.

num_args Specifies the number of entries in the argument list.

The XtCreateWidget function performs all the boilerplate operations of widget creation, doing
the following in order:

- Checks to see if the class initialize procedure has been called for this class and for all
superclasses and, if not, calls those necessary in a superclass-to-subclass order.
- If the specified class is not coreWidgetClass or a subclass thereof, and the parent’s
class is a subclass of compositeWidgetClass and either no extension record in the
parent’s composite class part extension field exists with the record_type NULLQUARK
or the accepts _objects field in the extension record is False, XtCreateWidget issues a
fatal error; see Section 3.1 and Chapter 12.
- Allocates memory for the widget instance.
- If the parent is a member of the class constraintWidgetClass, allocates memory for the
parent’s constraints and stores the address of this memory into the constraints
field.
- Initializes the Core nonresource data fields (for example, parent and visible).
- Initializes the resource fields (for example, background_pixel) by using the Core-
ClassPart resource lists specified for this class and all superclasses.
- If the parent is a member of the class constraintWidgetClass, initializes the resource
fields of the constraints record by using the ConstraintClassPart resource lists specified
for the parent’s class and all superclasses up to constraintWidgetClass.
- Calls the initialize procedures for the widget starting at the Object initialize procedure on
down to the widget’s initialize procedure.
- If the parent is a member of the class compositeWidgetClass, puts the widget into its
parent’s children list by calling its parent’s insert_child procedure. For further informa-
tion, see Section 3.1.
- If the parent is a member of the class constraintWidgetClass, calls the Con-
straintClassPart initialize procedures, starting at constraintWidgetClass on down to
the parent’s ConstraintClassPart initialize procedure.

To create an instance of a widget using varargs lists, use XtVaCreateWidget.

Widget XtVaCreateWidget(name, object_class, parent, ...)
    String name;
    WidgetClass object_class;
    Widget parent;

name Specifies the resource name for the created widget.
object_class

Specifies the widget class pointer for the created object. Must be objectClass or any subclass thereof.

parent

Specifies the parent widget. Must be of class Object or any subclass thereof.

... 

Specifies the variable argument list to override any other resource specifications.

The XtVaCreateWidget procedure is identical in function to XtCreateWidget with the args and num_args parameters replaced by a varargs list, as described in Section 2.5.1.

2.5.3. Creating an Application Shell Instance

An application can have multiple top-level widgets, each of which specifies a unique widget tree which can potentially be on different screens or displays. An application uses XtAppCreateShell to create independent widget trees.

Widget XtAppCreateShell(name, application_class, widget_class, display, args, num_args)

String name;
String application_class;
WidgetClass widget_class;
Display *display;
ArgList args;
Cardinal num_args;

name

Specifies the instance name of the shell widget. If name is NULL, the application name passed to XtDisplayInitialize is used.

application_class

Specifies the resource class string to be used in place of the widget class string when widget_class is applicationShellWidgetClass or a subclass thereof.

widget_class

Specifies the widget class for the top-level widget (e.g., applicationShellWidgetClass)

display

Specifies the display for the default screen and for the resource database used to retrieve the shell widget resources.

args

Specifies the argument list to override any other resource specifications.

num_args

Specifies the number of entries in the argument list.

The XtAppCreateShell function creates a new shell widget instance as the root of a widget tree. The screen resource for this widget is determined by first scanning args for the XtNscren argument. If no XtNscren argument is found, the resource database associated with the default screen of the specified display is queried for the resource name.screen, class Class. Screen where Class is the specified application_class if widget_class is applicationShellWidgetClass or a subclass thereof. If widget_class is not applicationShellWidgetClass or a subclass, Class is the class_name field from the CoreClassPart of the specified widget_class. If this query fails, the default screen of the specified display is used. Once the screen is determined, the resource database associated with that screen is used to retrieve all remaining resources for the shell widget not specified in args. The widget name and Class as determined above are used as the leftmost (i.e., root) components in all fully qualified resource names for objects within this widget tree.

If the specified widget class is a subclass of WMShell, the name and Class as determined above will be stored into the WM_CLASS property on the widget’s window when it becomes realized. If the specified widget_class is applicationShellWidgetClass or a subclass thereof the WM_COMMAND property will also be set from the values of the XtNargv and XtNargc.
To create multiple top-level shells within a single (logical) application, you can use one of two methods:

- Designate one shell as the real top-level shell and create the others as pop-up children of it by using XtCreatePopupShell.
- Have all shells as pop-up children of an unrealized top-level shell.

The first method, which is best used when there is a clear choice for what is the main window, leads to resource specifications like the following:

```plaintext
xmail.geometry:... (the main window)
xmail.read.geometry:... (the read window)
xmail.compose.geometry:... (the compose window)
```

The second method, which is best if there is no main window, leads to resource specifications like the following:

```plaintext
xmail.headers.geometry:... (the headers window)
xmail.read.geometry:... (the read window)
xmail.compose.geometry:... (the compose window)
```

To create a top-level widget that is the root of a widget tree using varargs lists, use XtVaAppCreateShell.

```c
Widget XtVaAppCreateShell(name, application_class, widget_class, display, ...)
```

- `name`: Specifies the instance name of the shell widget. If `name` is NULL, the application name passed to XtDisplayInitialize is used.
- `application_class`: Specifies the resource class string to be used in place of the widget class name string when `widget_class` is `applicationShellWidgetClass` or a subclass thereof.
- `widget_class`: Specifies the widget class for the top-level widget.
- `display`: Specifies the display for the default screen and for the resource database used to retrieve the shell widget resources.
- `...`: Specifies the variable argument list to override any other resource specifications.

The XtVaAppCreateShell procedure is identical in function to XtAppCreateShell with the `args` and `num_args` parameters replaced by a varargs list, as described in Section 2.5.1.

### 2.5.4. Convenience Procedure to Initialize an Application

To initialize the Intrinsics internals, create an application context, open and initialize a display, and create the initial application shell instance, an application may use XtAppInitialize or XtVaAppInitialize.
Widget XtAppInitialize(app_context_return, application_class, options, num_options, argc_in_out, argv_in_out, fallback_resources, args, num_args)

XtAppContext *app_context_return;
String application_class;
XrmOptionDescList options;
Cardinal num_options;
int *argc_in_out;
String *argv_in_out;
String *fallback_resources;
ArgList args;
Cardinal num_args;

app_context_return Returns the application context, if non-NULL.
application_class Specifies the class name of the application.
options Specifies the command line options table.
um_options Specifies the number of entries in options.
argc_in_out Specifies a pointer to the number of command line arguments.
argv_in_out Specifies a pointer to the command line arguments.
fallback_resources Specifies resource values to be used if the application class resource file cannot be opened or read, or NULL.
args Specifies the argument list to override any other resource specifications for the created shell widget.
um_args Specifies the number of entries in the argument list.

The XtAppInitialize function calls XtToolkitInitialize followed by XtCreateApplicationContext, then calls XtOpenDisplay with display_string NULL and application_name NULL, and finally calls XtAppCreateShell with application_name NULL, widget_class applicationShellWidgetClass, and the specified args and num_args and returns the created shell. The modified argc and argv returned by XtDisplayInitialize are returned in argc_in_out and argv_in_out. If app_context_return is not NULL, the created application context is also returned. If the display specified by the command line cannot be opened, an error message is issued and XtAppInitialize terminates the application. If fallback_resources is non-NULL, XtAppSetFallbackResources is called with the value prior to calling XtOpenDisplay.

Widget XtVaAppInitialize(app_context_return, application_class, options, num_options, argc_in_out, argv_in_out, fallback_resources, ...)

XtAppContext *app_context_return;
String application_class;
XrmOptionDescList options;
Cardinal num_options;
int *argc_in_out;
String *argv_in_out;
String *fallback_resources;

app_context_return Returns the application context, if non-NULL.
application_class Specifies the class name of the application.
options Specifies the command line options table.
um_options Specifies the number of entries in options.
argc_in_out Specifies a pointer to the number of command line arguments.
argv_in_out Specifies the command line arguments array.
 fallback_resources  Specifies resource values to be used if the application class resource
file cannot be opened, or NULL.

...  Specifies the variable argument list to override any other resource
specifications for the created shell.

The XtVaAppInitialize procedure is identical in function to XtAppInitialize with the args
and num_args parameters replaced by a varargs list, as described in Section 2.5.1.

2.5.5. Widget Instance Initialization: the initialize Procedure
The initialize procedure pointer in a widget class is of type XtInitProc.

typedef void (*XtInitProc)(Widget, Widget, ArgList, Cardinal*);
    Widget request;
    Widget new;
    ArgList args;
    Cardinal *num_args;

request  Specifies a copy of the widget with resource values as requested by the argu¬
ment list, the resource database, and the widget defaults.

new  Specifies the widget with the new values, both resource and nonresource, that
are actually allowed.

args  Specifies the argument list passed by the client, for computing derived resource
values. If the client created the widget using a varargs form, any resources
specified via XtVaTypedArg are converted to the widget representation and
the list is transformed into the ArgList format.

num_args  Specifies the number of entries in the argument list.

An initialization procedure performs the following:

• Allocates space for and copies any resources referenced by address that the client is
allowed to free or modify after the widget has been created. For example, if a widget
has a field that is a String, it may choose not to depend on the characters at that address
remaining constant but dynamically allocate space for the string and copy it to the new
space. Widgets that do not copy one or more resources referenced by address should
clearly so state in their user documentation.

    Note
    It is not necessary to allocate space for or to copy callback lists.

• Computes values for unspecified resource fields. For example, if width and height are
zero, the widget should compute an appropriate width and height based on its other
resources.

    Note
    A widget may only directly assign its own width and height within the ini¬
tialize, initialize_hook, set_values and set_values_hook procedures; see
Chapter 6.

• Computes values for uninitialized nonresource fields that are derived from resource
fields. For example, graphics contexts (GCs) that the widget uses are derived from
resources like background, foreground, and font.

An initialization procedure also can check certain fields for internal consistency. For example,
it makes no sense to specify a colormap for a depth that does not support that colormap.
Initialization procedures are called in superclass-to-subclass order after all fields specified in the resource lists have been initialized. The initialize procedure does not need to examine args and num_args if all public resources are declared in the resource list. Most of the initialization code for a specific widget class deals with fields defined in that class and not with fields defined in its superclasses.

If a subclass does not need an initialization procedure because it does not need to perform any of the above operations, it can specify NULL for the initialize field in the class record.

Sometimes a subclass may want to overwrite values filled in by its superclass. In particular, size calculations of a superclass are often incorrect for a subclass, and in this case, the subclass must modify or recalculate fields declared and computed by its superclass.

As an example, a subclass can visually surround its superclass display. In this case, the width and height calculated by the superclass initialize procedure are too small and need to be incremented by the size of the surround. The subclass needs to know if its superclass’s size was calculated by the superclass or was specified explicitly. All widgets must place themselves into whatever size is explicitly given, but they should compute a reasonable size if no size is requested.

The request and new arguments provide the necessary information for a subclass to determine the difference between an explicitly specified field and a field computed by a superclass. The request widget is a copy of the widget as initialized by the arglist and resource database. The new widget starts with the values in the request, but it has been updated by all superclass initialization procedures called so far. A subclass initialize procedure can compare these two to resolve any potential conflicts.

In the above example, the subclass with the visual surround can see if the width and height in the request widget are zero. If so, it adds its surround size to the width and height fields in the new widget. If not, it must make do with the size originally specified.

The new widget will become the actual widget instance record. Therefore, the initialization procedure should do all its work on the new widget; the request widget should never be modified. If the initialize procedure needs to call any routines that operate on a widget, it should specify new as the widget instance.

2.5.6. Constraint Instance Initialization: the ConstraintClassPart initialize Procedure

The constraint initialization procedure pointer, found in the ConstraintClassPart initialize field of the widget class record, is of type XtlnitProc. The values passed to the parent constraint initialization procedures are the same as those passed to the child’s class widget initialization procedures.

The constraints field of the request widget points to a copy of the constraints record as initialized by the arglist and resource database.

The constraint initialization procedure should compute any constraint fields derived from constraint resources. It can make further changes to the new widget to make the widget and any other constraint fields conform to the specified constraints, for example, changing the widget’s size or position.

If a constraint class does not need a constraint initialization procedure, it can specify NULL for the initialize field of the ConstraintClassPart in the class record.

2.5.7. Nonwidget Data Initialization: the initialize_hook Procedure
Note

The initialize_hook procedure is obsolete, as the same information is now available to the initialize procedure. The procedure has been retained for those widgets that used it in previous releases.

The initialize_hook procedure pointer is of type XtArgsProc:

typedef void (*XtArgsProc)(Widget, ArgList, Cardinal*);

    Widget w;
    ArgList args;
    Cardinal *num_args;

    w Specifies the widget.
    args Specifies the argument list passed by the client. If the client created the widget using a varargs form, any resources specified via XtVaTypedArg are converted to the widget representation and the list is transformed into the ArgList format.
    num_args Specifies the number of entries in the argument list.

If this procedure is not NULL, it is called immediately after the corresponding initialize procedure or in its place if the initialize field is NULL.

The initialize_hook procedure allows a widget instance to initialize nonresource data using information from the specified argument list as if it were a resource.

2.6. Realizing Widgets

To realize a widget instance, use XtRealizeWidget.

void XtRealizeWidget(w)

    Widget w;

    w Specifies the widget. Must be of class Core or any subclass thereof.

If the widget is already realized, XtRealizeWidget simply returns. Otherwise it performs the following:

- Binds all action names in the widget’s translation table to procedures (see Section 10.1.2).
- Makes a postorder traversal of the widget tree rooted at the specified widget and calls each non-NULL change_managed procedure of all composite widgets that have one or more managed children.
- Constructs an XSetWindowAttributes structure filled in with information derived from the Core widget fields and calls the realize procedure for the widget, which adds any widget-specific attributes and creates the X window.
- If the widget is not a subclass of compositeWidgetClass, XtRealizeWidget returns; otherwise it continues and performs the following:
  - Descends recursively to each of the widget’s managed children and calls the realize procedures. Primitive widgets that instantiate children are responsible for realizing those children themselves.
  - Maps all of the managed children windows that have mapped_whenManaged True. If a widget is managed but mapped_when_managed is False, the widget is allocated visual space but is not displayed.

If the widget is a top-level shell widget (that is, it has no parent), and mapped_when_managed is True, XtRealizeWidget maps the widget window.
X Toolkit Intrinsics

XtCreateWidget, XtVaCreateWidget, XtRealizeWidget, XtManageChildren, XtUnmanageChildren, XtUnrealizeWidget, XtSetMappedWhenManaged, and XtDestroyWidget maintain the following invariants:

- If a composite widget is realized, then all its managed children are realized.
- If a composite widget is realized, then all its managed children that have \texttt{mapped\_when\_managed} \texttt{True} are mapped.

All Intrinsics functions and all widget routines should accept either realized or unrealized widgets. When calling the realize or change\_managed procedures for children of a composite widget, \texttt{XtRealizeWidget} calls the procedures in reverse order of appearance in the \texttt{Composite-Part children} list. By default, this ordering of the realize procedures will result in the stacking order of any newly created subwindows being top-to-bottom in the order of appearance on the list, and the most recently created child will be at the bottom.

To check whether or not a widget has been realized, use \texttt{XtIsRealized}.

\begin{verbatim}
Boolean XtIsRealized(w)
      Widget w;

  w  Specifies the widget. Must be of class Object or any subclass thereof.

The \texttt{XtIsRealized} function returns \texttt{True} if the widget has been realized, that is, if the widget has a nonzero window ID. If the specified object is not a widget, the state of the nearest widget ancestor is returned.

Some widget procedures (for example, set\_values) might wish to operate differently after the widget has been realized.
\end{verbatim}

\subsection*{2.6.1. Widget Instance Window Creation: the realize Procedure}

The realize procedure pointer in a widget class is of type \texttt{XtRealizeProc}.

\begin{verbatim}
typedef void (*XtRealizeProc)(Widget, XtValueMask*, XSetWindowAttributes*);

  Widget w;
  XtValueMask *value_mask;
  XSetWindowAttributes *attributes;

  w  Specifies the widget.

  value_mask  Specifies which fields in the \texttt{attributes} structure are used.

  attributes  Specifies the window attributes to use in the \texttt{XCreateWindow} call.
\end{verbatim}

The realize procedure must create the widget's window.

Before calling the class realize procedure, the generic \texttt{XtRealizeWidget} function fills in a mask and a corresponding \texttt{XSetWindowAttributes} structure. It sets the following fields in \texttt{attributes} and corresponding bits in \texttt{value\_mask} based on information in the widget core structure:

- The \texttt{background\_pixelmap} (or \texttt{background\_pixel} if \texttt{background\_pixelmap} is \texttt{XtUnspecifiedPixmap}) is filled in from the corresponding field.
- The \texttt{border\_pixelmap} (or \texttt{border\_pixel} if \texttt{border\_pixelmap} is \texttt{XtUnspecifiedPixmap}) is filled in from the corresponding field.
- The \texttt{colormap} is filled in from the corresponding field.
- The \texttt{event\_mask} is filled in based on the event handlers registered, the event translations specified, whether the \texttt{expose} field is non-NULL, and whether \texttt{visible\_interest} is True.
- The \texttt{bit\_gravity} is set to \texttt{NorthWestGravity} if the \texttt{expose} field is NULL.

These or any other fields in \texttt{attributes} and the corresponding bits in \texttt{value\_mask} can be set by the realize procedure.
Note that because realize is not a chained operation, the widget class realize procedure must update the XSetWindowAttributes structure with all the appropriate fields from non-Core superclasses.

A widget class can inherit its realize procedure from its superclass during class initialization. The realize procedure defined for coreWidgetClass calls XtCreateWindow with the passed value_mask and attributes and with window_class and visual set to CopyFromParent. Both compositeWidgetClass and constraintWidgetClass inherit this realize procedure, and most new widget subclasses can do the same (see Section 1.6.10).

The most common noninherited realize procedures set bit_gravity in the mask and attributes to the appropriate value and then create the window. For example, depending on its justification, Label might set bit_gravity to WestGravity, CenterGravity, or EastGravity. Consequently, shrinking it would just move the bits appropriately, and no exposure event is needed for repainting.

If a composite widget's children should be realized in an order other than that specified (to control the stacking order, for example), it should call XtRealizeWidget on its children itself in the appropriate order from within its own realize procedure.

Widgets that have children and whose class is not a subclass of compositeWidgetClass are responsible for calling XtRealizeWidget on their children, usually from within the realize procedure.

2.6.2. Window Creation Convenience Routine
Rather than call the Xlib XCreateWindow function explicitly, a realize procedure should normally call the Intrinsics analog XtCreateWindow, which simplifies the creation of windows for widgets.

```c
void XtCreateWindow(w, window_class, visual, value_mask, attributes)
  Widget w;
  unsigned int window_class;
  Visual *visual;
  XtValueMask value_mask;
  XSetWindowAttributes *attributes;

  Specifies the widget that defines the additional window attributed. Must be of class Core or any subclass thereof.

  Specifies the Xlib window class (for example, InputOutput, InputOnly, or CopyFromParent).

  Specifies the visual type (usually CopyFromParent).

  Specifies which fields in the attributes structure are used.

  Specifies the window attributes to use in the XCreateWindow call.
```

The XtCreateWindow function calls the Xlib XCreateWindow function with values from the widget structure and the passed parameters. Then, it assigns the created window to the widget's window field.

XtCreateWindow evaluates the following fields of the widget core structure: depth, screen, parent->core.window, x, y, width, height, and border_width.

2.7. Obtaining Window Information from a Widget
The Core widget class definition contains the screen and window ids. The window field may be NULL for a while (see Sections 2.5 and 2.6).

The display pointer, the parent widget, screen pointer, and window of a widget are available to the widget writer by means of macros and to the application writer by means of functions.
Display *XtDisplay(w)
   Widget w;

w Specifies the widget. Must be of class Core or any subclass thereof.
XtDisplay returns the display pointer for the specified widget.

Widget XtParent(w)
   Widget w;

w Specifies the widget. Must be of class Object or any subclass thereof.
XtParent returns the parent object for the specified widget. The returned object will be of
class Object or a subclass.

Screen *XtScreen(w)
   Widget w;

w Specifies the widget. Must be of class Core or any subclass thereof.
XtScreen returns the screen pointer for the specified widget.

Window XtWindow(w)
   Widget w;

w Specifies the widget. Must be of class Core or any subclass thereof.
XtWindow returns the window of the specified widget.

The display pointer, screen pointer, and window of a widget or of the closest widget ancestor
of a nonwidget object are available by means of XtDisplayOfObject, XtScreenOfObject, and
XtWindowOfObject.

Display *XtDisplayOfObject(object)
   Widget object;

object Specifies the object. Must be of class Object or any subclass thereof.
XtDisplayOfObject is identical in function to XtDisplay if the object is a widget; otherwise
XtDisplayOfObject returns the display pointer for the nearest ancestor of object that is of
class Widget or a subclass thereof.

Screen *XtScreenOfObject(object)
   Widget object;

object Specifies the object. Must be of class Object or any subclass thereof.
XtScreenOfObject is identical in function to XtScreen if the object is a widget; otherwise
XtScreenOfObject returns the screen pointer for the nearest ancestor of object that is of class
Widget or a subclass thereof.

Window XtWindowOfObject(object)
   Widget object;

object Specifies the object. Must be of class Object or any subclass thereof.
XtWindowOfObject is identical in function to XtWindow if the object is a widget; otherwise
XtWindowOfObject returns the window for the nearest ancestor of object that is of class
Widget or a subclass thereof.
To retrieve the instance name of an object, use `XtName`.

```c
String XtName(object)
    Widget object;
```

`object` specifies the object whose name is desired. Must be of class Object or any subclass thereof.

`XtName` returns a pointer to the instance name of the specified object. The storage is owned by the Intrinsics and must not be modified. The name is not qualified by the names of any of the object's ancestors.

Several window attributes are locally cached in the widget instance. Thus, they can be set by the resource manager and `XtSetValues` as well as used by routines that derive structures from these values (for example, `depth` for deriving pixmaps, `background_pixel` for deriving GCs, and so on) or in the `XtCreateWindow` call.

The `x`, `y`, `width`, `height`, and `border_width` window attributes are available to geometry managers. These fields are maintained synchronously inside the Intrinsics. When an `XConfigureWindow` is issued by the Intrinsics on the widget's window (on request of its parent), these values are updated immediately rather than some time later when the server generates a `ConfigureNotify` event. (In fact, most widgets do not select `SubstructureNotify` events.) This ensures that all geometry calculations are based on the internally consistent toolkit world rather than on either an inconsistent world updated by asynchronous `ConfigureNotify` events or a consistent but slow world in which geometry managers ask the server for window sizes whenever they need to lay out their managed children (see Chapter 6).

### 2.7.1. Unrealizing Widgets

To destroy the windows associated with a widget and its non-pop-up descendants, use `XtUnrealizeWidget`.

```c
void XtUnrealizeWidget(w)
    Widget w;
```

`w` specifies the widget. Must be of class Core or any subclass thereof.

If the widget is currently unrealized, `XtUnrealizeWidget` simply returns. Otherwise it performs the following:

- Unmanages the widget if the widget is managed.
- Makes a postorder (child-to-parent) traversal of the widget tree rooted at the specified widget and, for each widget that has declared a callback list resource named “unrealizeCallback”, executes the procedures on the `XtNunrcaiizeCallback` list.
- Destroys the widget’s window and any subwindows by calling `XDestroyWindow` with the specified widget’s `window` field.

Any events in the queue or which arrive following a call to `XtUnrealizeWidget` will be dispatched as if the window(s) of the unrealized widget(s) had never existed.

### 2.8. Destroying Widgets

The Intrinsics provide support:

- To destroy all the pop-up children of the widget being destroyed and destroy all children of composite widgets.
- To remove (and unmap) the widget from its parent.
- To call the callback procedures that have been registered to trigger when the widget is destroyed.
• To minimize the number of things a widget has to deallocate when destroyed.
• To minimize the number of XDestroyWindow calls when destroying a widget tree.

To destroy a widget instance, use XtDestroyWidget.

```c
void XtDestroyWidget(w);
```

`w` Specifies the widget. Must be of class Object or any subclass thereof.

The `XtDestroyWidget` function provides the only method of destroying a widget, including widgets that need to destroy themselves. It can be called at any time, including from an application callback routine of the widget being destroyed. This requires a two-phase destroy process in order to avoid dangling references to destroyed widgets.

In phase 1, `XtDestroyWidget` performs the following:

- If the `being_destroyed` field of the widget is `True`, it returns immediately.
- Recursively descends the widget tree and sets the `being_destroyed` field to `True` for the widget and all normal and pop-up children.
- Adds the widget to a list of widgets (the destroy list) that should be destroyed when it is safe to do so.

Entries on the destroy list satisfy the invariant that if `w2` occurs after `w1` on the destroy list, then `w2` is not a descendent, either normal or pop-up, of `w1`.

Phase 2 occurs when all procedures that should execute as a result of the current event have been called, including all procedures registered with the event and translation managers, that is, when the current invocation of `XtDispatchEvent` is about to return, or immediately if not in `XtDispatchEvent`.

In phase 2, `XtDestroyWidget` performs the following on each entry in the destroy list in the order specified:

- Calls the destroy callback procedures registered on the widget and all normal and pop-up descendants in postorder (it calls child callbacks before parent callbacks).
- If the widget is not a pop-up child and the widget's parent is a subclass of `composite-WidgetClass`, and if the parent is not being destroyed, it calls `XtUnmanageChild` on the widget and then calls the widget's parent's delete_child procedure (see Section 3.3).
- If the widget is not a pop-up child and the widget's parent is a subclass of `constraint-WidgetClass`, it calls the `ConstraintClassPart` destroy procedure for the parent, then for the parent's superclass, until finally it calls the `ConstraintClassPart` destroy procedure for `constraintWidgetClass`.
- Calls the destroy procedures for the widget and all normal and pop-up descendants in postorder. For each such widget, it calls the `CoreClassPart` destroy procedure declared in the widget class, then the destroy procedure declared in its superclass, until finally it calls the destroy procedure declared in the Object class record.
- Calls `XDestroyWindow` if the specified widget is realized (that is, has an X window). The server recursively destroys all normal descendant windows.
- Recursively descends the tree and destroys the windows for all realized pop-up descendants, deallocates all pop-up descendants, constraint records, callback lists, and if the widget's class is a subclass of `compositeWidgetClass`, children.

### 2.8.1. Adding and Removing Destroy Callbacks

When an application needs to perform additional processing during the destruction of a widget, it should register a destroy callback procedure for the widget. The destroy callback procedures use the mechanism described in Chapter 8. The destroy callback list is identified by the
resource name XtNdestroyCallback.
For example, the following adds an application-supplied destroy callback procedure ClientDest-

troy with client data to a widget by calling XtAddCallback.

\[
\text{XtAddCallback}(w, \text{XtNdestroyCallback}, \text{ClientDestroy}, \text{client\_data})
\]

Similarly, the following removes the application-supplied destroy callback procedure ClientDestroy by calling XtRemoveCallback.

\[
\text{XtRemoveCallback}(w, \text{XtNdestroyCallback}, \text{ClientDestroy}, \text{client\_data})
\]

The ClientDestroy argument is of type XtCallbackProc; see Section 8.1.

2.8.2. Dynamic Data Deallocation: the destroy Procedure
The destroy procedure pointers in the ObjectClassPart, RectObjClassPart, and Core-
ClassPart structures are of type XtWidgetProc.

typedef void (*XtWidgetProc)(Widget);

\[
\text{Widget} w;
\]

\[
\text{w} \quad \text{Specifies the widget being destroyed.}
\]

The destroy procedures are called in subclass-to-superclass order. Therefore, a widget's des-
stroy procedure only should deallocate storage that is specific to the subclass and should ignore
the storage allocated by any of its superclasses. The destroy procedure should only deallocate
resources that have been explicitly created by the subclass. Any resource that was obtained
from the resource database or passed in an argument list was not created by the widget and
therefore should not be destroyed by it. If a widget does not need to deallocate any storage,
the destroy procedure entry in its class record can be NULL.

Deallocating storage includes, but is not limited to, the following steps:

- Calling XtFree on dynamic storage allocated with XtMalloc, XtCalloc, and so on.
- Calling XFreePixmap on pixmaps created with direct X calls.
- Calling XtReleaseGC on GCs allocated with XtGetGC.
- Calling XFreeGC on GCs allocated with direct X calls.
- Calling XtRemoveEventHandler on event handlers added to other widgets.
- Calling XtRemoveTimeOut on timers created with XtAppAddTimeOut.
- Calling XtDestroyWidget for each child if the widget has children and is not a subclass
  of compositeWidgetClass.

During destroy phase 2 for each widget, the Intrinsics remove the widget from the modal cas-
cade, unregister all event handlers, remove all key, keyboard, button, and pointer grabs and
remove all callback procedures registered on the widget. Any outstanding selection transfers
will time out.

2.8.3. Dynamic Constraint Data Deallocation: the ConstraintClassPart destroy Procedure
The constraint destroy procedure identified in the ConstraintClassPart structure is called for a
widget whose parent is a subclass of constraintWidgetClass. This constraint destroy pro-
duce pointer is of type XtWidgetProc. The constraint destroy procedures are called in
subclass-to-superclass order, starting at the class of the widget's parent and ending at
constraintWidgetClass. Therefore, a parent's constraint destroy procedure only should deal-
locate storage that is specific to the constraint subclass and not storage allocated by any of its
superclasses.
If a parent does not need to deallocate any constraint storage, the constraint destroy procedure entry in its class record can be NULL.

2.9. Exiting from an Application
All X Toolkit applications should terminate by calling XtDestroyApplicationContext and then exiting using the standard method for their operating system (typically, by calling exit for POSIX-based systems). The quickest way to make the windows disappear while exiting is to call XtUnmapWindow on each top-level shell widget. The Intrinsics have no resources beyond those in the program image, and the X server will free its resources when its connection to the application is broken.

Depending upon the widget set in use, it may be necessary to explicitly destroy individual widgets or widget trees with XtDestroyWidget before calling XtDestroyApplicationContext in order to ensure that any required widget cleanup is properly executed. The application developer must refer to the widget documentation to learn if a widget needs to perform additional cleanup beyond that performed automatically by the operating system. None of the widget classes defined by the Intrinsics require additional cleanup.
Chapter 3
Composite Widgets and Their Children

Composite widgets (widgets whose class is a subclass of compositeWidgetClass) can have an arbitrary number of children. Consequently, they are responsible for much more than primitive widgets. Their responsibilities (either implemented directly by the widget class or indirectly by Intrinsics functions) include

- Overall management of children from creation to destruction.
- Destruction of descendants when the composite widget is destroyed.
- Physical arrangement (geometry management) of a displayable subset of children (that is, the managed children).
- Mapping and unmapping of a subset of the managed children.

Overall management is handled by the generic procedures XtCreateWidget and XtDestroyWidget. XtCreateWidget adds children to their parent by calling the parent’s insert_child procedure. XtDestroyWidget removes children from their parent by calling the parent’s delete_child procedure and ensures that all children of a destroyed composite widget also get destroyed.

Only a subset of the total number of children is actually managed by the geometry manager and hence possibly visible. For example, a composite editor widget supporting multiple editing buffers might allocate one child widget for each file buffer, but it might only display a small number of the existing buffers. Widgets that are in this displayable subset are called managed widgets and enter into geometry manager calculations. The other children are called unmanaged widgets and, by definition, are not mapped by the Intrinsics.

Children are added to and removed from their parent’s managed set by using XtManageChild, XtManageChildren, XtUnmanageChild, and XtUnmanageChildren, which notify the parent to recalculate the physical layout of its children by calling the parent’s change_managed procedure. The XtCreateManagedWidget convenience function calls XtCreateWidget and XtManageChild on the result.

Most managed children are mapped, but some widgets can be in a state where they take up physical space but do not show anything. Managed widgets are not mapped automatically if their map_when_managed field is False. The default is True and is changed by using XtSetMappedWhenManaged.

Each composite widget class declares a geometry manager, which is responsible for figuring out where the managed children should appear within the composite widget’s window. Geometry management techniques fall into four classes:

**Fixed boxes**
Fixed boxes have a fixed number of children created by the parent. All these children are managed, and none ever makes geometry manager requests.

**Homogeneous boxes**
Homogeneous boxes treat all children equally and apply the same geometry constraints to each child. Many clients insert and delete widgets freely.

**Heterogeneous boxes**
Heterogeneous boxes have a specific location where each child is placed. This location usually is not specified in pixels, because the window may be resized, but is expressed rather in terms of the relationship between a child and the parent or between the child and other specific children. The class of heterogeneous boxes is usually a subclass of Constraint.
Shell boxes typically have only one child, and the child’s size is usually exactly the size of the shell. The geometry manager must communicate with the window manager, if it exists, and the box must also accept ConfigureNotify events when the window size is changed by the window manager.

3.1. Addition of Children to a Composite Widget: the insert_child Procedure

To add a child to the parent’s list of children, the XtCreateWidget function calls the parent’s class routine insert_child. The insert_child procedure pointer in a composite widget is of type XtWidgetProc.

typedef void (*XtWidgetProc)(Widget);

    Widget w;

    w Passes the newly created child.

Most composite widgets inherit their superclass’s operation. The insert_child routine in CompositeWidgetClass calls and inserts the child at the specified position in the children list, expanding it if necessary.

Some composite widgets define their own insert_child routine so that they can order their children in some convenient way, create companion controller widgets for a new widget, or limit the number or class of their child widgets. A composite widget class that wishes to allow nonwidget children (see Chapter 12) must specify a CompositeClassExtension extension record as described in section 1.4.2.1 and set the accepts_objects field in this record to True. If the CompositeClassExtension record is not specified or the accepts_objects field is False, the composite widget can assume that all its children are of a subclass of Core without an explicit subclass test in the insert_child procedure.

If there is not enough room to insert a new child in the children array (that is, num_children is equal to num_slots), the insert_child procedure must first reallocate the array and update num_slots. The insert_child procedure then places the child at the appropriate position in the array and increments the num_children field.

3.2. Insertion Order of Children: the insert_position Procedure

Instances of composite widgets sometimes need to specify more about the order in which their children are kept. For example, an application may want a set of command buttons in some logical order grouped by function, and it may want buttons that represent file names to be kept in alphabetical order without constraining the order in which the buttons are created.

An application controls the presentation order of a set of children by supplying an XtNinsertPosition resource. The insert_position procedure pointer in a composite widget instance is of type XtOrderProc.

typedef Cardinal (*XtOrderProc)(Widget);

    Widget w;

    w Passes the newly created widget.

Composite widgets that allow clients to order their children (usually homogeneous boxes) can call their widget instance’s insert_position procedure from the class’s insert_child procedure to determine where a new child should go in its children array. Thus, a client using a composite class can apply different sorting criteria to widget instances of the class, passing in a different insert_position procedure resource when it creates each composite widget instance.

The return value of the insert_position procedure indicates how many children should go before the widget. Returning zero indicates that the widget should go before all other children, and returning num_children indicates that it should go after all other children. The default
3.3. Deletion of Children: the delete_child Procedure

To remove the child from the parent’s children list, the XtDestroyWidget function eventually causes a call to the Composite parent’s class delete_child procedure. The delete_child procedure pointer is of type XtWidgetProc.

typedef void (*XtWidgetProc)(Widget);

w Passes the child being deleted.

Most widgets inherit the delete_child procedure from their superclass. Composite widgets that create companion widgets define their own delete_child procedure to remove these companion widgets.

3.4. Adding and Removing Children from the Managed Set

The Intrinsics provide a set of generic routines to permit the addition of widgets to or the removal of widgets from a composite widget’s managed set. These generic routines eventually call the composite widget’s change_managed procedure if the procedure pointer is non-NULL. The change_managed procedure pointer is of type XtWidgetProc. The widget argument specifies the composite widget whose managed child set has been modified.

3.4.1. Managing Children

To add a list of widgets to the geometry-managed (and hence displayable) subset of their Composite parent, use XtManageChildren.

typedef Widget *WidgetList;

void XtManageChildren(children, num_children)
    WidgetList children;
    Cardinal num_children;

children Specifies a list of child widgets. Each child must be of class RectObj or any subclass thereof.

num_children Specifies the number of children in the list.

The XtManageChildren function performs the following:

- Issues an error if the children do not all have the same parent or if the parent’s class is not a subclass of compositeWidgetClass.
- Returns immediately if the common parent is being destroyed; otherwise, for each unique child on the list, XtManageChildren ignores the child if it already is managed or is being destroyed, and marks it if not.
- If the parent is realized and after all children have been marked, it makes some of the newly managed children viewable:
  - Calls the change_managed routine of the widgets’ parent.
  - Calls XtRealizeWidget on each previously unmanaged child that is unrealized.
  - Maps each previously unmanaged child that has map_when_managed True.

Managing children is independent of the ordering of children and independent of creating and deleting children. The layout routine of the parent should consider children whose managed field is True and should ignore all other children. Note that some composite widgets,
especially fixed boxes, call `XtManageChild` from their `insert_child` procedure. If the parent widget is realized, its `change_managed` procedure is called to notify it that its set of managed children has changed. The parent can reposition and resize any of its children. It moves each child as needed by calling `XtMoveWidget`, which first updates the $x$ and $y$ fields and which then calls `XMoveWindow`. If the composite widget wishes to change the size or border width of any of its children, it calls `XtResizeWidget`, which first updates the `width`, `height`, and `border_width` fields and then calls `XConfigureWindow`. Simultaneous repositioning and resizing may be done with `XtConfigureWidget`; see Section 6.6.

To add a single child to its parent widget's set of managed children, use `XtManageChild`.

```c
void XtManageChild(Widget child);
```

`child` Specifies the child. Must be of class `RectObj` or any subclass thereof.

The `XtManageChild` function constructs a `WidgetList` of length 1 and calls `XtManageChildren`.

To create and manage a child widget in a single procedure, use `XtCreateManagedWidget` or `XtVaCreateManagedWidget`.

```c
Widget XtCreateManagedWidget(String name, WidgetClass widget_class, Widget parent, ArgList args, Cardinal num_args);
```

`name` Specifies the resource instance name for the created widget.

`widget_class` Specifies the widget class pointer for the created widget. Must be `RectObjClass` or any subclass thereof.

`parent` Specifies the parent widget. Must be of class `Composite` or any subclass thereof.

`args` Specifies the argument list to override any other resource specifications.

`num_args` Specifies the number of entries in the argument list.

The `XtCreateManagedWidget` function is a convenience routine that calls `XtCreateWidget` and `XtManageChild`.

```c
Widget XtVaCreateManagedWidget(String name, WidgetClass widget_class, Widget parent, ...);
```

`name` Specifies the resource instance name for the created widget.

`widget_class` Specifies the widget class pointer for the created widget. Must be `RectObjClass` or any subclass thereof.

`parent` Specifies the parent widget. Must be of class `Composite` or any subclass thereof.

`...` Specifies the variable argument list to override any other resource specifications.
XtVaCreateManagedWidget is identical in function to XtCreateManagedWidget with the args and num_args parameters replaced by a varargs list, as described in Section 2.5.1.

3.4.2. Unmanaging Children
To remove a list of children from a parent widget's managed list, use XtUnmanageChildren.

```c
void XtUnmanageChildren(children, num_children)
    WidgetList children;
    Cardinal num_children;
```

- `children` Specifies a list of child widgets. Each child must be of class RectObj or any subclass thereof.
- `num_children` Specifies the number of children.

The XtUnmanageChildren function performs the following:
- Issues an error if the children do not all have the same parent or if the parent is not a subclass of compositeWidgetClass.
- Returns immediately if the common parent is being destroyed; otherwise, for each unique child on the list, XtUnmanageChildren performs the following:
  - Ignores the child if it already is unmanaged or is being destroyed, and marks it if not.
  - If the child is realized, it makes it nonvisible by unmapping it.
- Calls the change_managed routine of the widgets' parent after all children have been marked if the parent is realized.

XtUnmanageChildren does not destroy the child widgets. Removing widgets from a parent’s managed set is often a temporary banishment, and some time later the client may manage the children again. To destroy widgets entirely, XtDestroyWidget should be called instead; see Section 2.9.

To remove a single child from its parent widget's managed set, use XtUnmanageChild.

```c
void XtUnmanageChild(child)
    Widget child;
```

- `child` Specifies the child. Must be of class RectObj or any subclass thereof.

The XtUnmanageChild function constructs a widget list of length 1 and calls XtUnmanageChildren.

These functions are low-level routines that are used by generic composite widget building routines. In addition, composite widgets can provide widget-specific, high-level convenience procedures.

3.4.3. Determining If a Widget Is Managed
To determine the managed state of a given child widget, use XtIsManaged.

```c
Boolean XtIsManaged(w)
    Widget w;
```

- `w` Specifies the widget. Must be of class Object or any subclass thereof.

The XtIsManaged function returns True if the specified widget is of class RectObj or any subclass thereof and is managed, or False otherwise.
3.5. Controlling When Widgets Get Mapped

A widget is normally mapped if it is managed. However, this behavior can be overridden by setting the `XtNmappedWhenManaged` resource for the widget when it is created or by setting the `map_when_managed` field to `False`.

To change the value of a given widget’s `map_when_managed` field, use `XtSetMappedWhenManaged`.

```c
void XtSetMappedWhenManaged(w, map_when_managed)
    Widget w;
    Boolean map_when_managed;
```

- `w` Specifies the widget. Must be of class Core or any subclass thereof.
- `map_when_managed` Specifies a Boolean value that indicates the new value that is stored into the widget’s `map_when_managed` field.

If the widget is realized and managed and if `map_when_managed` is `True`, `XtSetMappedWhenManaged` maps the window. If the widget is realized and managed and if `map_when_managed` is `False`, it unmaps the window. `XtSetMappedWhenManaged` is a convenience function that is equivalent to (but slightly faster than) calling `XtSetValues` and setting the new value for the `XtNmappedWhenManaged` resource then mapping the widget as appropriate. As an alternative to using `XtSetMappedWhenManaged` to control mapping, a client may set `mapped_when_managed` to `False` and use `XtMapWidget` and `XtUnmapWidget` explicitly.

To map a widget explicitly, use `XtMapWidget`.

```c
XtMapWidget(w)
    Widget w;
```

- `w` Specifies the widget. Must be of class Core or any subclass thereof.

To unmap a widget explicitly, use `XtUnmapWidget`.

```c
XtUnmapWidget(w)
    Widget w;
```

- `w` Specifies the widget. Must be of class Core or any subclass thereof.

3.6. Constrained Composite Widgets

The `Constraint` widget class is a subclass of `compositeWidgetClass`. The name is derived from the fact that constraint widgets may manage the geometry of their children based on constraints associated with each child. These constraints can be as simple as the maximum width and height the parent will allow the child to occupy or can be as complicated as how other children should change if this child is moved or resized. Constraint widgets let a parent define constraints as resources that are supplied for their children. For example, if the Constraint parent defines the maximum sizes for its children, these new size resources are retrieved for each child as if they were resources that were defined by the child widget’s class. Accordingly, constraint resources may be included in the argument list or resource file just like any other resource for the child.

Constraint widgets have all the responsibilities of normal composite widgets and, in addition, must process and act upon the constraint information associated with each of their children.

To make it easy for widgets and the Intrinsics to keep track of the constraints associated with a child, every widget has a `constraints` field, which is the address of a parent-specific structure.
that contains constraint information about the child. If a child’s parent does not belong to a subclass of `constraintWidgetClass`, then the child’s `constraints` field is NULL.

Subclasses of Constraint can add constraint data to the constraint record defined by their superclass. To allow this, widget writers should define the constraint records in their private .h file by using the same conventions as used for widget records. For example, a widget class that needs to maintain a maximum width and height for each child might define its constraint record as follows:

```c
typedef struct {
    Dimension max_width, max_height;
} MaxConstraintPart;

typedef struct {
    MaxConstraintPart max;
} MaxConstraintRecord, *MaxConstraint;
```

A subclass of this widget class that also needs to maintain a minimum size would define its constraint record as follows:

```c
typedef struct {
    Dimension min_width, min_height;
} MinConstraintPart;

typedef struct {
    MaxConstraintPart max;
    MinConstraintPart min;
} MaxMinConstraintRecord, *MaxMinConstraint;
```

Constraints are allocated, initialized, deallocated, and otherwise maintained insofar as possible by the Intrinsics. The Constraint class record part has several entries that facilitate this. All entries in `ConstraintClassPart` are fields and procedures that are defined and implemented by the parent, but they are called whenever actions are performed on the parent’s children.

The `XtCreateWidget` function uses the `constraint_size` field in the parent’s class record to allocate a constraint record when a child is created. `XtCreateWidget` also uses the constraint resources to fill in resource fields in the constraint record associated with a child. It then calls the constraint initialize procedure so that the parent can compute constraint fields that are derived from constraint resources and can possibly move or resize the child to conform to the given constraints.

When the `XtGetValues` and `XtSetValues` functions are executed on a child, they use the constraint resources to get the values or set the values of constraints associated with that child. `XtSetValues` then calls the constraint set_values procedures so that the parent can recompute derived constraint fields and move or resize the child as appropriate. If a Constraint widget class or any of its superclasses have declared a `ConstraintClassExtension` record in the `ConstraintClassPart extension` fields with a record type of `NULLQUARK` and the `get_values_hook` field in the extension record is non-NULL, `XtGetValues` calls the `get_values_hook` procedure(s) to allow the parent to return derived constraint fields.

The `XtDestroyWidget` function calls the constraint destroy procedure to deallocate any dynamic storage associated with a constraint record. The constraint record itself must not be deallocated by the constraint destroy procedure; `XtDestroyWidget` does this automatically.
Chapter 4

Shell Widgets

Shell widgets hold an application's top-level widgets to allow them to communicate with the window manager. Shells have been designed to be as nearly invisible as possible. Clients have to create them, but they should never have to worry about their sizes.

If a shell widget is resized from the outside (typically by a window manager), the shell widget also resizes its managed child widget automatically. Similarly, if the shell's child widget needs to change size, it can make a geometry request to the shell, and the shell negotiates the size change with the outer environment. Clients should never attempt to change the size of their shells directly.

The four types of public shells are:

- OverrideShell: Used for shell windows that completely bypass the window manager (for example, pop-up menu shells).
- TransientShell: Used for shell windows that have the WM_TRANSIENT_FOR property set. The effect of this property is dependent upon the window manager being used.
- TopLevelShell: Used for normal top-level windows (for example, any additional top-level widgets an application needs).
- ApplicationShell: Used for the single main top-level window that the window manager identifies as an application instance and that interacts with the session manager.

4.1. Shell Widget Definitions

Widgets negotiate their size and position with their parent widget, that is, the widget that directly contains them. Widgets at the top of the hierarchy do not have parent widgets. Instead, they must deal with the outside world. To provide for this, each top-level widget is encapsulated in a special widget, called a shell widget.

Shell widgets, whose class is a subclass of the Composite class, encapsulate other widgets and can allow a widget to avoid the geometry clipping imposed by the parent-child window relationship. They also can provide a layer of communication with the window manager.

The seven different types of shells are:

- Shell: The base class for shell widgets; provides the fields needed for all types of shells. Shell is a direct subclass of compositeWidgetClass.
- OverrideShell: A subclass of Shell; used for shell windows that completely bypass the window manager.
- WMShell: A subclass of Shell; contains fields needed by the common window manager protocol.
- VendorShell: A subclass of WMShell; contains fields used by vendor-specific window managers.
TransientShell  A subclass of VendorShell; used for shell windows that desire the WM_TRANSIENT_FOR property.

TopLevelShell  A subclass of VendorShell; used for normal top level windows.

ApplicationShell  A subclass of TopLevelShell; used for an application's main top-level window.

Note that the classes Shell, WMShell, and VendorShell are internal and should not be instantiated or subclassed. Only OverrideShell, TransientShell, TopLevelShell, and ApplicationShell are intended for public use.

4.1.1. ShellClassPart Definitions

Only the Shell class has additional class fields, which are all contained in the ShellClassExtensionRec. None of the other Shell classes have any additional class fields:

typedef struct { XtPointer extension; } ShellClassPart, OverrideShellClassPart, WMShellClassPart, VendorShellClassPart, TransientShellClassPart, TopLevelShellClassPart, ApplicationShellClassPart;

The full Shell class record definitions are

typedef struct __ShellClassRec {
  CoreClassPart core_class;
  CompositeClassPart composite_class;
  ShellClassPart shell_class;
} ShellClassRec;

typedef struct {
  XtPointer next_extension;
  XrmQuark record_type;
  long version;
  Cardinal record_size;
  XtGeometryHandler root_geometry_manager;
} ShellClassExtensionRec, *ShellClassExtension;

typedef struct __OverrideShellClassRec {
  CoreClassPart core_class;
  CompositeClassPart composite_class;
  ShellClassPart shell_class;
  OverrideShellClassPart override_shell_class;
} OverrideShellClassRec;

typedef struct __WMShellClassRec {
  CoreClassPart core_class;
  CompositeClassPart composite_class;
  ShellClassPart shell_class;
  WMShellClassPart wm_shell_class;
} WMShellClassRec;
typedef struct _VendorShellClassRec {
    CoreClassPart core_class;
    CompositeClassPart composite_class;
    ShellClassPart shell_class;
    WMShellClassPart wm_shell_class;
    VendorShellClassPart vendor_shell_class;
} VendorShellClassRec;

typedef struct _TransientShellClassRec {
    CoreClassPart core_class;
    CompositeClassPart composite_class;
    ShellClassPart shell_class;
    WMShellClassPart wm_shell_class;
    VendorShellClassPart vendor_shell_class;
    TransientShellClassPart transient_shell_class;
} TransientShellClassRec;

typedef struct _TopLevelShellClassRec {
    CoreClassPart core_class;
    CompositeClassPart composite_class;
    ShellClassPart shell_class;
    WMShellClassPart wm_shell_class;
    VendorShellClassPart vendor_shell_class;
    TopLevelShellClassPart top_level_shell_class;
} TopLevelShellClassRec;

typedef struct _ApplicationShellClassRec {
    CoreClassPart core_class;
    CompositeClassPart composite_class;
    ShellClassPart shell_class;
    WMShellClassPart wm_shell_class;
    VendorShellClassPart vendor_shell_class;
    TopLevelShellClassPart top_level_shell_class;
    ApplicationShellClassPart application_shell_class;
} ApplicationShellClassRec;

The single occurrences of the class records and pointers for creating instances of shells are

extern ShellClassRec shellClassRec;
extern OverrideShellClassRec overrideShellClassRec;
extern WMShellClassRec wmShellClassRec;
extern VendorShellClassRec vendorShellClassRec;
extern TransientShellClassRec transientShellClassRec;
extern TopLevelShellClassRec topLevelShellClassRec;
extern ApplicationShellClassRec applicationShellClassRec;

extern WidgetClass shellWidgetClass;
extern WidgetClass overrideShellWidgetClass;
extern WidgetClass wmShellWidgetClass;
extern WidgetClass vendorShellWidgetClass;
extern WidgetClass transientShellWidgetClass;
extern WidgetClass topLevelShellWidgetClass;
extern WidgetClass applicationShellWidgetClass;

The following opaque types and opaque variables are defined for generic operations on widgets
whose class is a subclass of Shell.
### Types

<table>
<thead>
<tr>
<th>Types</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShellWidget</td>
<td>shellWidgetClass</td>
</tr>
<tr>
<td>OverrideShellWidget</td>
<td>overrideShellWidgetClass</td>
</tr>
<tr>
<td>WMShellWidget</td>
<td>wmShellWidgetClass</td>
</tr>
<tr>
<td>VendorShellWidget</td>
<td>vendorShellWidgetClass</td>
</tr>
<tr>
<td>TransientShellWidget</td>
<td>transientShellWidgetClass</td>
</tr>
<tr>
<td>TopLevelShellWidget</td>
<td>topLevelShellWidgetClass</td>
</tr>
<tr>
<td>ApplicationShellWidget</td>
<td>applicationShellWidgetClass</td>
</tr>
<tr>
<td>ShellWidgetClass</td>
<td></td>
</tr>
<tr>
<td>OverrideShellWidgetClass</td>
<td></td>
</tr>
<tr>
<td>WMShellWidgetClass</td>
<td></td>
</tr>
<tr>
<td>VendorShellWidgetClass</td>
<td></td>
</tr>
<tr>
<td>TransientShellWidgetClass</td>
<td></td>
</tr>
<tr>
<td>TopLevelShellWidgetClass</td>
<td></td>
</tr>
<tr>
<td>ApplicationShellWidgetClass</td>
<td></td>
</tr>
</tbody>
</table>

The declarations for all Intrinsics-defined shells except VendorShell appear in Shell.h and ShellP.h. VendorShell has separate public and private .h files which are included by Shell.h and ShellP.h.

Shell.h uses incomplete structure definitions to ensure that the compiler catches attempts to access private data in any of the Shell instance or class data structures.

The symbolic constant for the ShellClassExtension version identifier is XtShellExtensionVersion (see Section 1.6.12).

The root_geometry_manager procedure acts as the parent geometry manager for geometry requests made by shell widgets. When a shell widget calls either XtMakeGeometryRequest or XtMakeResizeRequest, the root_geometry_manager procedure is invoked to negotiate the new geometry with the window manager. If the window manager permits the new geometry, the root_geometry_manager procedure should return XtGeometryYes; if the window manager denies the geometry request or it does not change the window geometry within some timeout interval (equal to wm_timeout in the case of WMShells), the root_geometry_manager procedure should return XtGeometryNo. If the window manager makes some alternative geometry change, the root_geometry_manager procedure may either return XtGeometryNo and handle the new geometry as a resize, or may return XtGeometryAlmost in anticipation that the shell will accept the compromise. If the compromise is not accepted, the new size must then be handled as a resize. Subclasses of Shell that wish to provide their own root_geometry_manager procedures are strongly encouraged to use enveloping to invoke their superclass’s root_geometry_manager procedure under most situations, as the window manager interaction may be very complex.

If no ShellClassPart extension record is declared with record_type equal to NULLQUARK, then XtInheritRootGeometryManager is assumed.

### 4.1.2. ShellPart Definition

The various shell widgets have the following additional instance fields defined in their widget records:
typedef struct {
    String geometry;
    XtCreatePopupChildProc create_popup_child_proc;
    XtGrabKind grab_kind;
    Boolean spring_loaded;
    Boolean popped_up;
    Boolean allow_shell_resize;
    Boolean client_specified;
    Boolean save_under;
    Boolean override_redirect;
    XtCallbackList popup_callback;
    XtCallbackList popdown_callback;
    Visual* visual;
} ShellPart;

typedef struct { int empty; } OverrideShellPart;

typedef struct {
    String title;
    int wm_timeout;
    Boolean wait_for_wm;
    Boolean transient;
    struct _OldXSizeHints {
        long flags;
        int x, y;
        int width, height;
        int min_width, min_height;
        int max_width, max_height;
        int width_inc, height_inc;
        struct {
            int x;
            int y;
        } min_aspect, max_aspect;
    } size_hints;
    XWMHints wm_hints;
    int base_width, base_height, win_gravity;
    Atom title_encoding;
} WMSheUPart;

typedef struct {
    int vendor_specific;
} VendorshellPart;

typedef struct {
    Widget transient_for;
} TransientShellPart;

typedef struct {
    String icon_name;
    Boolean iconic;
    Atom icon_name_encoding;
} TopLevelShellPart;
typedef struct {
    char *class;
    XrmClass xrm_class;
    int argc;
    char **argv;
} ApplicationShellPart;

The full shell widget instance record definitions are

typedef struct {
    CorePart core;
    CompositePart composite;
    ShellPart shell;
} ShellRec, *ShellWidget;

typedef struct {
    CorePart core;
    CompositePart composite;
    ShellPart shell;
    OverrideShellPart override;
} OverrideShellRec, *OverrideShellWidget;

typedef struct {
    CorePart core;
    CompositePart composite;
    ShellPart shell;
    WMShellPart wm;
} WMShellRec, *WMShellWidget;

typedef struct {
    CorePart core;
    CompositePart composite;
    ShellPart shell;
    WMShellPart wm;
    VendorShellPart vendor;
} VendorShellRec, *VendorShellWidget;

typedef struct {
    CorePart core;
    CompositePart composite;
    ShellPart shell;
    WMShellPart wm;
    VendorShellPart vendor;
    TransientShellPart transient;
} TransientShellRec, *TransientShellWidget;

typedef struct {
    CorePart core;
    CompositePart composite;
    ShellPart shell;
    WMShellPart wm;
    VendorShellPart vendor;
    TopLevelShellPart topLevel;
} TopLevelShellRec, *TopLevelShellWidget;
typedef struct {
    CorePart core;
    CompositePart composite;
    ShellPart shell;
    WMShellPart wm;
    VendorShellPart vendor;
    TopLevelShellPart topLevel;
    ApplicationShellPart application;
} ApplicationShellRec, *ApplicationShellWidget;

4.1.3. Shell Resources

The resource names, classes, and representation types specified in the shellClassRec resource list are

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNallowShellResize</td>
<td>XtCAllowShellResize</td>
<td>XtRBoolean</td>
</tr>
<tr>
<td>XtNcreatePopupChildProc</td>
<td>XtCCreatePopupChildProc</td>
<td>XtRFunction</td>
</tr>
<tr>
<td>XtNgeometry</td>
<td>XtCGeometry</td>
<td>XtRString</td>
</tr>
<tr>
<td>XtNoverrideRedirect</td>
<td>XtCOverrideRedirect</td>
<td>XtRBoolean</td>
</tr>
<tr>
<td>XtNpopdownCallback</td>
<td>XtCCallback</td>
<td>XtRCallback</td>
</tr>
<tr>
<td>XtNpopupCallback</td>
<td>XtCCallback</td>
<td>XtRCallback</td>
</tr>
<tr>
<td>XtNsaveUnder</td>
<td>XtCSaveUnder</td>
<td>XtRBoolean</td>
</tr>
<tr>
<td>XtNvisual</td>
<td>XtCVisual</td>
<td>XtRVisual</td>
</tr>
</tbody>
</table>

OverrideShell declares no additional resources beyond those defined by Shell.

The resource names, classes, and representation types specified in the wmShellClassRec resource list are

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNbaseHeight</td>
<td>XtCBaseHeight</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNbaseWidth</td>
<td>XtCBaseWidth</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNheightInc</td>
<td>XtCHeightInc</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNiconMask</td>
<td>XtCIconMask</td>
<td>XtRBitmap</td>
</tr>
<tr>
<td>XtNiconPixmap</td>
<td>XtCIconPixmap</td>
<td>XtRBitmap</td>
</tr>
<tr>
<td>XtNiconWindow</td>
<td>XtCIconWindow</td>
<td>XtRWindow</td>
</tr>
<tr>
<td>XtNiconX</td>
<td>XtCIconX</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNiconY</td>
<td>XtCIconY</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNinitialState</td>
<td>XtCInitialState</td>
<td>XtRInitialState</td>
</tr>
<tr>
<td>XtNinput</td>
<td>XtCInput</td>
<td>XtRBool</td>
</tr>
<tr>
<td>XtNmaxAspectX</td>
<td>XtCMaxAspectX</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNmaxAspectY</td>
<td>XtCMaxAspectY</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNmaxHeight</td>
<td>XtCMaxHeight</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNmaxWidth</td>
<td>XtCMaxWidth</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNminAspectX</td>
<td>XtCMinAspectX</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNminAspectY</td>
<td>XtCMinAspectY</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNminHeight</td>
<td>XtCMinHeight</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNminWidth</td>
<td>XtCMinWidth</td>
<td>XtRInt</td>
</tr>
<tr>
<td>XtNtitle</td>
<td>XtCTitle</td>
<td>XtRString</td>
</tr>
<tr>
<td>XtNtitleEncoding</td>
<td>XtCTitleEncoding</td>
<td>XtRAtom</td>
</tr>
</tbody>
</table>
The class resource list for VendorShell is implementation-defined.

The resource names, classes, and representation types that are specified in the transient-ShellClassRec resource list are

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xt transientFor</td>
<td>Xt CT transientFor</td>
<td>Xt R Widget</td>
</tr>
</tbody>
</table>

The resource names, classes, and representation types that are specified in the topLevelShellClassRec resource list are

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNiconName</td>
<td>Xt CIconName</td>
<td>Xt R String</td>
</tr>
<tr>
<td>XtNiconNameEncoding</td>
<td>Xt CIconNameEncoding</td>
<td>Xt R Atom</td>
</tr>
<tr>
<td>XtNiconic</td>
<td>Xt CIconic</td>
<td>Xt R Boolean</td>
</tr>
</tbody>
</table>

The resource names, classes, and representation types that are specified in the application-ShellClassRec resource list are

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNargc</td>
<td>Xt CArgc</td>
<td>Xt R Int</td>
</tr>
<tr>
<td>XtNargv</td>
<td>Xt CArgv</td>
<td>Xt R StringArray</td>
</tr>
</tbody>
</table>

### 4.1.4. ShellPart Default Values

The default values for fields common to all classes of public shells (filled in by the Shell resource lists and the Shell initialize procedures) are

<table>
<thead>
<tr>
<th>Field</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometry</td>
<td>NULL</td>
</tr>
<tr>
<td>create_popup_child_proc</td>
<td>NULL</td>
</tr>
<tr>
<td>grab_kind</td>
<td>(none)</td>
</tr>
<tr>
<td>spring_loaded</td>
<td>(none)</td>
</tr>
<tr>
<td>popped_up</td>
<td>False</td>
</tr>
<tr>
<td>allow_shell_resize</td>
<td>False</td>
</tr>
<tr>
<td>client_specified</td>
<td>(internal)</td>
</tr>
<tr>
<td>save_under</td>
<td>True for OverrideShell and TransientShell, False otherwise</td>
</tr>
<tr>
<td>override_redirect</td>
<td>True for OverrideShell, False otherwise</td>
</tr>
<tr>
<td>popup_callback</td>
<td>NULL</td>
</tr>
</tbody>
</table>
The `geometry` field specifies the size and position and is usually given only on a command line or in a defaults file. If the `geometry` field is non-NULL when a widget of class WMShell is realized, the geometry specification is parsed using `XWMGeometry` with a default geometry string constructed from the values of `x`, `y`, `width`, `height`, `width_inc`, and `height_inc` and the size and position flags in the window manager size hints are set. If the geometry specifies an `x` or `y` position, then `USPosition` is set. If the geometry specifies a `width` or `height`, then `USSize` is set. Any fields in the geometry specification override the corresponding values in the Core `x`, `y`, `width`, and `height` fields. If `geometry` is NULL or contains only a partial specification, then the Core `x`, `y`, `width`, and `height` fields are used and `PPosition` and `PSize` are set as appropriate. The geometry string is not copied by any of the Intrinsics Shell classes; a client specifying the string in an arglist or varargs list must ensure that the value remains valid until the shell widget is realized. For further information on the geometry string, see Section 10.3 in `Xlib - C Language X Interface`.

The `create_popup_child_proc` procedure is called by the `XtPopup` procedure and may remain NULL. The `grab_kind`, `spring_loaded`, and `popped_up` fields maintain widget state information as described under `XtPopup`, `XtMenuPopup`, `XtPopdown`, and `XtMenuPopdown`. The `allow_shell_resize` field controls whether the widget contained by the shell is allowed to try to resize itself. If `allow_shell_resize` is False, any geometry requests made by the child will always return `XtGeometryNo` without interacting with the window manager. Setting `save_under` True instructs the server to attempt to save the contents of windows obscured by the shell when it is mapped and to restore those contents automatically when the shell is unmapped. It is useful for pop-up menus. Setting `override_redirect` True determines whether the window manager can intercede when the shell window is mapped. The pop-up and pop-down callbacks are called during `XtPopup` and `XtPopdown`. For further information on `override_redirect`, see Section 3.2 in `Xlib - C Language X Interface` and Sections 4.1.10 and 4.2.2 in the `Inter-Client Communication Conventions Manual`.

The default values for Shell fields in WMShell and its subclasses are

<table>
<thead>
<tr>
<th>Field</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Icon name, if specified, otherwise the application’s name.</td>
</tr>
<tr>
<td>wm_timeout</td>
<td>Five seconds, in units of milliseconds.</td>
</tr>
<tr>
<td>transient</td>
<td>True</td>
</tr>
<tr>
<td>min_width</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>min_height</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>max_width</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>max_height</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>width_inc</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>height_inc</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>min_aspect_x</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>min_aspect_y</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>max_aspect_x</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>max_aspect_y</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>input</td>
<td>False</td>
</tr>
<tr>
<td>initial_state</td>
<td>Normal</td>
</tr>
<tr>
<td>icon_pixmap</td>
<td>None</td>
</tr>
<tr>
<td>icon_window</td>
<td>None</td>
</tr>
<tr>
<td>icon_x</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
<tr>
<td>icon_y</td>
<td><code>XtUnspecifiedShellInt</code></td>
</tr>
</tbody>
</table>
The `title` and `title_encoding` fields are stored in the `WM_NAME` property on the shell’s window by the WMShell realize procedure. If the `title_encoding` field is `None`, the `title` string is assumed to be in the encoding of the current locale and the encoding of the `WM_NAME` property is set to `XStdICCTextStyle`. If a language procedure has not been set the default value of `title_encoding` is `XA_STRING`, otherwise the default value is `None`. The `wm_timeout` field specifies, in milliseconds, the amount of time a shell is to wait for confirmation of a geometry request to the window manager. If none comes back within that time, the shell assumes the window manager is not functioning properly and sets `wait_for_wm` to `False` (later events may reset this value). When `wait_for_wm` is `False`, the shell does not wait for a response but relies on asynchronous notification. If `transient` is `True`, the `WM_TRANSIENT_FOR` property will be stored on the shell window with a value as specified below. The interpretation of this property is specific to the window manager under which the application is run; see the Inter-Client Communication Conventions Manual for more details. All other resources specify fields in the window manager hints and the window manager size hints. The realize and `set_values` procedures of WMShell set the corresponding flag bits in the hints if any of the fields contain non-default values. In addition, if a flag bit is set that refers to a field with the value `XtUnspecifiedShellInt`, the value of the field is modified as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>base_width, base_height</code></td>
<td>0</td>
</tr>
<tr>
<td><code>width_inc, height_inc</code></td>
<td>1</td>
</tr>
<tr>
<td><code>max_width, max_height</code></td>
<td>32767</td>
</tr>
<tr>
<td><code>min_width, min_height</code></td>
<td>1</td>
</tr>
<tr>
<td><code>min_aspect_x, min_aspect_y</code></td>
<td>-1</td>
</tr>
<tr>
<td><code>max_aspect_x, max_aspect_y</code></td>
<td>-1</td>
</tr>
<tr>
<td><code>icon_x, icon_y</code></td>
<td>-1</td>
</tr>
<tr>
<td><code>win_gravity</code></td>
<td>value returned by <code>XWMGeometry</code> if called, else <code>NorthWestGravity</code></td>
</tr>
</tbody>
</table>

If the shell widget has a non-NULL parent, then the realize and `set_values` procedures replace the value `XtUnspecifiedWindow` in the `window_group` field with the window id of the root widget of the widget tree if the root widget is realized. The symbolic constant `XtUnspecifiedWindowGroup` may be used to indicate that the `window_group` hint flag bit is not to be set. If `transient` is `True` and the shell’s class is not a subclass of TransientShell and `window_group` is not `XtUnspecifiedWindowGroup` the WMShell realize and `set_values` procedures then store the `WM_TRANSIENT_FOR` property with the value of `window_group`.

Transient shells have the following additional resource:

<table>
<thead>
<tr>
<th>Field</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>transient_for</td>
<td>NULL</td>
</tr>
</tbody>
</table>
The realize and set_values procedures of TransientShell store the WM_TRANSIENT_FOR property on the shell window if transient is True. If transient_for is non-NULL and the widget specified by transient_for is realized, then its window is used as the value of the WM_TRANSIENT_FOR property; otherwise, the value of window_group is used.

TopLevel shells have the following additional resources:

<table>
<thead>
<tr>
<th>Field</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>icon_name</td>
<td>Shell widget’s name</td>
</tr>
<tr>
<td>iconic</td>
<td>False</td>
</tr>
<tr>
<td>icon_name_encoding</td>
<td>See text</td>
</tr>
</tbody>
</table>

The icon_name and icon_name_encoding fields are stored in the WM_ICON_NAME property on the shell’s window by the TopLevelShell realize procedure. If the icon_name_encoding field is None, the icon_name string is assumed to be in the encoding of the current locale and the encoding of the WM_ICON_NAME property is set to XStdICCTextStyle. If a language procedure has not been set the default value of icon_name_encoding is XA_STRING, otherwise the default value is None. The iconic field may be used by a client to request that the window manager iconify or deiconify the shell; the TopLevelShell set_values procedure will send the appropriate WM_CHANGE_STATE message (as specified by the Inter-Client Communication Conventions Manual) if this resource is changed from False to True, and will call XtPopup specifying grab_kind as XtGrabNone if iconic is changed from True to False. The XtNiconic resource is also an alternative way to set the XtNinitialState resource to indicate that a shell should be initially displayed as an icon; the TopLevelShell initialize procedure will set initial_state to IconicState if iconic is True.

Application shells have the following additional resources:

<table>
<thead>
<tr>
<th>Field</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>argc</td>
<td>0</td>
</tr>
<tr>
<td>argv</td>
<td>NULL</td>
</tr>
</tbody>
</table>

The argc and argv fields are used to initialize the standard property WM_COMMAND. See the Inter-Client Communication Conventions Manual for more information.
Chapter 5

Pop-Up Widgets

Pop-up widgets are used to create windows outside of the window hierarchy defined by the widget tree. Each pop-up child has a window that is a descendant of the root window, so that the pop-up window is not clipped by the pop-up widget's parent window. Therefore, pop-ups are created and attached differently to their widget parent than normal widget children.

A parent of a pop-up widget does not actively manage its pop-up children; in fact, it usually does not operate upon them in any way. The *popup_list* field in the CorePart structure contains the list of its pop-up children. This pop-up list exists mainly to provide the proper place in the widget hierarchy for the pop-up to get resources and to provide a place for XtDestroyWidget to look for all extant children.

A composite widget can have both normal and pop-up children. A pop-up can be popped up from almost anywhere, not just by its parent. The term *child* always refers to a normal, geometry-managed widget on the composite widget's list of children, and the term *pop-up child* always refers to a widget on the pop-up list.

5.1. Pop-Up Widget Types

There are three kinds of pop-up widgets:

- **Modeless pop-ups**
  A modeless pop-up (for example, a dialog box that does not prevent continued interaction with the rest of the application) can usually be manipulated by the window manager and looks like any other application window from the user's point of view. The application main window itself is a special case of a modeless pop-up.

- **Modal pop-ups**
  A modal pop-up (for example, a dialog box that requires user input to continue) can sometimes be manipulated by the window manager, and except for events that occur in the dialog box, it disables user-event distribution to the rest of the application.

- **Spring-loaded pop-ups**
  A spring-loaded pop-up (for example, a menu) can seldom be manipulated by the window manager, and except for events that occur in the pop-up or its descendants, it disables user-event distribution to all other applications.

Modal pop-ups and spring-loaded pop-ups are very similar and should be coded as if they were the same. In fact, the same widget (for example, a ButtonBox or Menu widget) can be used both as a modal pop-up and as a spring-loaded pop-up within the same application. The main difference is that spring-loaded pop-ups are brought up with the pointer and, because of the grab that the pointer button causes, require different processing by the Intrinsic. Further, all user input remap events occurring outside the spring-loaded pop-up (e.g., in a descendant) are also delivered to the spring-loaded pop-up after they have been dispatched to the appropriate descendant, so that, for example, button-down can take down a spring-loaded pop-up no matter where the button-down occurs.

Any kind of pop-up, in turn, can pop up other widgets. Modal and spring-loaded pop-ups can constrain user events to the most recent such pop-up or allow user events to be dispatched to any of the modal or spring-loaded pop-ups currently mapped.

Regardless of their type, all pop-up widget classes are responsible for communicating with the X window manager and therefore are subclasses of one of the Shell widget classes.
5.2. Creating a Pop-Up Shell

For a widget to be popped up, it must be the child of a pop-up shell widget. None of the Intrinsic-supplied shells will simultaneously manage more than one child. Both the shell and child taken together are referred to as the pop-up. When you need to use a pop-up, you always refer to the pop-up by the pop-up shell, not the child.

To create a pop-up shell, use `XtCreatePopupShell`.

```c
Widget XtCreatePopupShell(String name, WidgetClass widget_class, Widget parent, ArgList args, Cardinal num_args)
```

- `name` Specifies the instance name for the created shell widget.
- `widget_class` Specifies the widget class pointer for the created shell widget.
- `parent` Specifies the parent widget. Must be of class Core or any subclass thereof.
- `args` Specifies the argument list to override any other resource specifications.
- `num_args` Specifies the number of entries in the argument list.

The `XtCreatePopupShell` function ensures that the specified class is a subclass of Shell and, rather than using `insert_child` to attach the widget to the parent's children list, attaches the shell to the parent's `popup_list` directly.

The screen resource for this widget is determined by first scanning `args` for the `XtNscreen` argument. If no `XtNscreen` argument is found, the resource database associated with the parent’s screen is queried for the resource `name.scrccn`, class `Class.SctQen` where `Class` is the `class_name` field from the `CoreClassPart` of the specified `widget_class`. If this query fails, the parent’s screen is used. Once the screen is determined, the resource database associated with that screen is used to retrieve all remaining resources for the widget not specified in `args`.

A spring-loaded pop-up invoked from a translation table via `XtMenuPopup` must already exist at the time that the translation is invoked, so the translation manager can find the shell by name. Pop-ups invoked in other ways can be created when the pop-up actually is needed. This delayed creation of the shell is particularly useful when you pop up an unspecified number of pop-ups. You can look to see if an appropriate unused shell (that is, not currently popped up) exists and create a new shell if needed.

To create a pop-up shell using varargs lists, use `XtVaCreatePopupShell`.

```c
Widget XtVaCreatePopupShell(String name, WidgetClass widget_class, Widget parent, ...
```

- `name` Specifies the instance name for the created shell widget.
- `widget_class` Specifies the widget class pointer for the created shell widget.
- `parent` Specifies the parent widget. Must be of class Core or any subclass thereof.
- `...` Specifies the variable argument list to override any other resource specifications.

`XtVaCreatePopupShell` is identical in function to `XtCreatePopupShell` with the `args` and `num_args` parameters replaced by a varargs list as described in Section 2.5.1.
5.3. Creating Pop-Up Children

Once a pop-up shell is created, the single child of the pop-up shell can be created either statically or dynamically.

At startup, an application can create the child of the pop-up shell, which is appropriate for pop-up children composed of a fixed set of widgets. The application can change the state of the subparts of the pop-up child as the application state changes. For example, if an application creates a static menu, it can call XtSetSensitive (or, in general, XtSetValues) on any of the buttons that make up the menu. Creating the pop-up child early means that pop-up time is minimized, especially if the application calls XtRealizeWidget on the pop-up shell at startup. When the menu is needed, all the widgets that make up the menu already exist and need only be mapped. The menu should pop up as quickly as the X server can respond.

Alternatively, an application can postpone the creation of the child until it is needed, which minimizes application startup time and allows the pop-up child to reconfigure itself each time it is popped up. In this case, the pop-up child creation routine might poll the application to find out if it should change the sensitivity of any of its subparts.

Pop-up child creation does not map the pop-up, even if you create the child and call XtRealizeWidget on the pop-up shell.

All shells have pop-up and pop-down callbacks, which provide the opportunity either to make last-minute changes to a pop-up child before it is popped up or to change it after it is popped down. Note that excessive use of pop-up callbacks can make popping up occur more slowly.

5.4. Mapping a Pop-Up Widget

Pop-ups can be popped up through several mechanisms:

- A call to XtPopup or XtPopupSpringLoaded.
- One of the supplied callback procedures XtCallbackNone, XtCallbackNonexclusive, or XtCallbackExclusive.
- The standard translation action XtMenuPopup.

Some of these routines take an argument of type XtGrabKind, which is defined as

typedef enum {XtGrabNone, XtGrabNonexclusive, XtGrabExclusive} XtGrabKind;

The create_popup_child_proc procedure pointer in the shell widget instance record is of type

XtCreatePopupChildProc.

typedef void (*)(XtCreatePopupChildProc)(Widget);

w Specifies the shell widget being popped up.

To map a pop-up from within an application, use XtPopup.

void XtPopup(popup_shell, grab_kind)
    Widget popup_shell;
    XtGrabKind grab_kind;

popup_shell Specifies the shell widget.

grab_kind Specifies the way in which user events should be constrained.

The XtPopup function performs the following:

- Calls XtCheckSubclass to ensure popup_shell's class is a subclass of shellWidgetClass.
• Raises the window and returns if the shell’s popped_up field is already True.
• Calls the callback procedures on the shell’s popup_callback list, specifying a pointer to the value of grab_kind as the call_data argument.
• Sets the shell popped_up field to True, the shell spring_loaded field to False, and the shell grab_kind field from grab_kind.
• If the shell’s create_popup_child_proc field is non-NULL, XtPopup calls it with popup_shell as the parameter.
• If grab_kind is either XtGrabNonexclusive or XtGrabExclusive, it calls
  XtAddGrab(popup_shell, (grab_kind == XtGrabExclusive), False)
• Calls XtRealizeWidget with popup_shell specified.
• Calls XMapRaised with the window of popup_shell.

To map a spring-loaded pop-up from within an application, use XtPopupSpringLoaded.

    void XtPopupSpringLoaded(popup_shell)
    
    popup_shell Specifies the shell widget to be popped up.

The XtPopupSpringLoaded function performs exactly as XtPopup except that it sets the shell spring_loaded field to True and always calls XtAddGrab with exclusive True and spring-loaded True.

To map a pop-up from a given widget’s callback list, you also can register one of the XtCallbackNone, XtCallbackNonexclusive, or XtCallbackExclusive convenience routines as callbacks, using the pop-up shell widget as the client data.

    void XtCallbackNone(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;

    w Specifies the widget.

    client_data Specifies the pop-up shell.

    call_data Specifies the callback data argument, which is not used by this procedure.

    void XtCallbackNonexclusive(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;

    w Specifies the widget.

    client_data Specifies the pop-up shell.

    call_data Specifies the callback data argument, which is not used by this procedure.

    void XtCallbackExclusive(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;

    w Specifies the widget.
client_data Specifies the pop-up shell.
call_data Specifies the callback data argument, which is not used by this procedure.

The XtCallbackNone, XtCallbackNonexclusive, and XtCallbackExclusive functions call XtPopup with the shell specified by the client_data argument and grab_kind set as the name specifies. XtCallbackNone, XtCallbackNonexclusive, and XtCallbackExclusive specify XtGrabNone, XtGrabNonexclusive, and XtGrabExclusive, respectively. Each function then sets the widget that executed the callback list to be insensitive by calling XtSetSensitive. Using these functions in callbacks is not required. In particular, an application must provide customized code for callbacks that create pop-up shells dynamically or that must do more than desensitizing the button.

Within a translation table, to pop up a menu when a key or pointer button is pressed or when the pointer is moved into a widget, use XtMenuPopup, or its synonym, MenuPopup. From a translation writer’s point of view, the definition for this translation action is

```c
void XtMenuPopup(shell_name)
    String shell_name;
```

shell_name Specifies the name of the shell widget to pop up.

XtMenuPopup is known to the translation manager, which registers the corresponding built-in action procedure XtMenuPopupAction using XtRegisterGrabAction specifying owner_events True, event_mask ButtonPressMask | ButtonReleaseMask, and pointer_mode and keyboard_mode GrabModeAsync.

If XtMenuPopup is invoked on ButtonPress, it calls XtPopupSpringLoaded on the specified shell widget. If XtMenuPopup is invoked on KeyPress or EnterWindow, it calls XtPopup on the specified shell widget with grab_kind set to XtGrabNonexclusive. Otherwise, the translation manager generates a warning message and ignores the action.

XtMenuPopup tries to find the shell by searching the widget tree starting at the widget in which it is invoked. If it finds a shell with the specified name in the pop-up children of that widget, it pops up the shell with the appropriate parameters. Otherwise, it moves up the parent chain to find a pop-up child with the specified name. If XtMenuPopup gets to the application top-level shell widget and has not found a matching shell, it generates a warning and returns immediately.

5.5. Unmapping a Pop-Up Widget

Pop-ups can be popped down through several mechanisms:

- A call to XtPopdown
- The supplied callback procedure XtCallbackPopdown
- The standard translation action XtMenuPopdown

To unmap a pop-up from within an application, use XtPopdown.

```c
void XtPopdown(popup_shell)
    Widget popup_shell;
```

popup_shell Specifies the shell widget to pop down.

The XtPopdown function performs the following:

- Calls XtCheckSubclass to ensure popup_shell’s class is a subclass of shellWidgetClass.
- Checks that the popped_up field of popup_shell is True; otherwise, it returns immediately.
Unmaps `popup_shell`'s window and, if `override redirect` is `False`, sends a synthetic `UnmapNotify` event as specified by the `Inter-Client Communication Conventions Manual`.

If `popup_shell`'s `grab_kind` is either `XtGrabNonexclusive` or `XtGrabExclusive`, it calls `XtRemoveGrab`.

Sets `popup_shell`'s `popped_up` field to `False`.

Calls the callback procedures on the shell's `popdown_callback` list, specifying a pointer to the value of the shell's `grab_kind` field as the `call_data` argument.

To pop down a pop-up from a callback list, you may use the callback `XtCallbackPopdown`:

```c
void XtCallbackPopdown(w, client_data, call_data)
    Widget w;
    XtPointer client_data;
    XtPointer call_data;

    w Specifies the widget.
    client_data Specifies a pointer to the `XtPopdownID` structure.
    call_data Specifies the callback data argument, which is not used by this procedure.
```

The `XtCallbackPopdown` function casts the `client_data` parameter to a pointer of type `XtPopdownID`.

```c
typedef struct {
    Widget shell_widget;
    Widget enable_widget;
} XtPopdownIDRec, *XtPopdownID;
```

The `shell_widget` is the pop-up shell to pop down, and the `enable_widget` is usually the widget that was used to pop it up in one of the pop-up callback convenience procedures.

`XtCallbackPopdown` calls `XtPopdown with the specified `shell_widget` and then calls `XtSetSensitive` to resensitize `enable_widget`.

Within a translation table, to pop down a spring-loaded menu when a key or pointer button is released or when the pointer is moved into a widget, use `XtMenuPopdown` or its synonym, `MenuPopdown`. From a translation writer's point of view, the definition for this translation action is

```c
void XtMenuPopdown(shell_name)
    String shell_name;
```

If a shell name is not given, `XtMenuPopdown` calls `XtPopdown with the widget for which the translation is specified. If `shell_name` is specified in the translation table, `XtMenuPopdown` tries to find the shell by looking up the widget tree starting at the widget in which it is invoked. If it finds a shell with the specified name in the pop-up children of that widget, it pops down the shell; otherwise, it moves up the parent chain to find a pop-up child with the specified name. If `XtMenuPopdown` gets to the application top-level shell widget and cannot find a matching shell, it generates a warning and returns immediately.
Chapter 6
Geometry Management

A widget does not directly control its size and location; rather, its parent is responsible for controlling them. Although the position of children is usually left up to their parent, the widgets themselves often have the best idea of their optimal sizes and, possibly, preferred locations.

To resolve physical layout conflicts between sibling widgets and between a widget and its parent, the Intrinsics provide the geometry management mechanism. Almost all composite widgets have a geometry manager specified in the `geometry_manager` field in the widget class record that is responsible for the size, position, and stacking order of the widget's children. The only exception is fixed boxes, which create their children themselves and can ensure that their children will never make a geometry request.

6.1. Initiating Geometry Changes

Parents, children, and clients each initiate geometry changes differently. Because a parent has absolute control of its children's geometry, it changes the geometry directly by calling `XtMoveWidget`, `XtResizeWidget`, or `XtConfigureWidget`. A child must ask its parent for a geometry change by calling `XtMakeGeometryRequest` or `XtMakeResizeRequest`. An application or other client code initiates a geometry change by calling `XtSetValues` on the appropriate geometry fields, thereby giving the widget the opportunity to modify or reject the client request before it gets propagated to the parent and the opportunity to respond appropriately to the parent's reply.

When a widget that needs to change its size, position, border width, or stacking depth asks its parent's geometry manager to make the desired changes, the geometry manager can allow the request, disallow the request, or suggest a compromise.

When the geometry manager is asked to change the geometry of a child, the geometry manager may also rearrange and resize any or all of the other children that it controls. The geometry manager can move children around freely using `XtMoveWidget`. When it resizes a child (that is, changes the width, height, or border width) other than the one making the request, it should do so by calling `XtResizeWidget`. The requesting child may be given special treatment; see Section 6.5. It can simultaneously move and resize a child with a single call to `XtConfigureWidget`.

Often, geometry managers find that they can satisfy a request only if they can reconfigure a widget that they are not in control of; in particular, the composite widget may want to change its own size. In this case, the geometry manager makes a request to its parent's geometry manager. Geometry requests can cascade this way to arbitrary depth.

Because such cascaded arbitration of widget geometry can involve extended negotiation, windows are not actually allocated to widgets at application startup until all widgets are satisfied with their geometry; see Sections 2.5 and 2.6.

Notes

1. The Intrinsics treatment of stacking requests is deficient in several areas. Stacking requests for unrealized widgets are granted but will have no effect. In addition, there is no way to do an `XtSetValues` that will generate a stacking geometry request.
2. After a successful geometry request (one that returned XtGeometryYes), a widget does not know whether its resize procedure has been called. Widgets should have resize procedures that can be called more than once without ill effects.

6.2. General Geometry Manager Requests
When making a geometry request, the child specifies an XtWidgetGeometry structure.

typedef unsigned long XtGeometryMask;

typedef struct {
    XtGeometryMask request_mode;
    Position x, y;
    Dimension width, height;
    Dimension border_width;
    Widget sibling;
    int stack_mode;
} XtWidgetGeometry;

To make a general geometry manager request from a widget, use XtMakeGeometryRequest.

XtGeometryResult XtMakeGeometryRequest(w, request, reply_return)

    Widget w;
    XtWidgetGeometry *request;
    XtWidgetGeometry *reply_return;

w Specifies the widget making the request. Must be of class RectObj or any subclass thereof.

request Specifies the desired widget geometry (size, position, border width, and stacking order).

reply_return Returns the allowed widget size, or may be NULL if the requesting widget is not interested in handling XtGeometryAlmost.

Depending on the condition, XtMakeGeometryRequest performs the following:

- If the widget is unmanaged or the widget’s parent is not realized, it makes the changes and returns XtGeometryYes.
- If the parent’s class is not a subclass of compositeWidgetClass or the parent’s geometry_manager field is NULL, it issues an error.
- If the widget’s being_destroyed field is True, it returns XtGeometryNo.
- If the widget x, y, width, height and, border_width fields are all equal to the requested values, it returns XtGeometryYes; otherwise, it calls the parent’s geometry_manager procedure with the given parameters.
- If the parent’s geometry manager returns XtGeometryYes and if XtCWQueryOnly is not set in request->request_mode and if the widget is realized, XtMakeGeometryRequest calls the XConfigureWindow Xlib function to reconfigure the widget’s window (set its size, location, and stacking order as appropriate).
- If the geometry manager returns XtGeometryDone, the change has been approved and actually has been done. In this case, XtMakeGeometryRequest does no configuring and returns XtGeometryYes. XtMakeGeometryRequest never returns XtGeometryDone.
- Otherwise, XtMakeGeometryRequest just returns the resulting value from the parent’s geometry manager.
Children of primitive widgets are always unmanaged; therefore, XtMakeGeometryRequest always returns XtGeometryYes when called by a child of a primitive widget.

The return codes from geometry managers are

typedef enum _XtGeometryResult {
    XtGeometryYes,
    XtGeometryNo,
    XtGeometryAlmost,
    XtGeometryDone
} XtGeometryResult;

The request_mode definitions are from <X11/X.h>.

#define CWX (1<<0)
#define CWY (1<<1)
#define CWWidth (1<<2)
#define CWHHeight (1<<3)
#define CWBorderWidth (1<<4)
#define CWSibling (1<<5)
#define CWStackMode (1<<6)

The Intrinsics also support the following value.

#define XtCWQueryOnly (1<<7)

XtCWQueryOnly indicates that the corresponding geometry request is only a query as to what would happen if this geometry request were made and that no widgets should actually be changed.

XtMakeGeometryRequest, like the XConfigureWindow Xlib function, uses request_mode to determine which fields in the XtWidgetGeometry structure the caller wants to specify.

The stack_mode definitions are from <X11/X.h>:

#define Above 0
#define Below 1
#define TopIf 2
#define BottomIf 3
#define Opposite 4

The Intrinsics also support the following value.

#define XtSMDontChange 5

For definition and behavior of Above, Below, TopIf, BottomIf, and Opposite, see Section 3.7 in Xlib – C Language X Interface. XtSMDontChange indicates that the widget wants its current stacking order preserved.

6.3. Resize Requests

To make a simple resize request from a widget, you can use XtMakeResizeRequest as an alternative to XtMakeGeometryRequest.

XtGeometryResult XtMakeResizeRequest(w, width, height, width_return, height_return)
    Widget w;
    Dimension width, height;
    Dimension *width_return, *height_return;

w Specifies the widget making the request. Must be of class RectObj or any subclass thereof.
X Toolkit Intrinsics

XI1 Release 5

width
height

Specify the desired widget width and height.

width_return
height_return

Return the allowed widget width and height.

The XtMakeResizeRequest function, a simple interface to XtMakeGeometryRequest, creates an XtWidgetGeometry structure and specifies that width and height should change by setting request mode to CWWidth | CWHeight. The geometry manager is free to modify any of the other window attributes (position or stacking order) to satisfy the resize request. If the return value is XtGeometryAlmost, width_return and height_return contain a compromise width and height. If these are acceptable, the widget should immediately call XtMakeResizeRequest again and request that the compromise width and height be applied. If the widget is not interested in XtGeometryAlmost replies, it can pass NULL for width_return and height_return.

6.4. Potential Geometry Changes

Sometimes a geometry manager cannot respond to a geometry request from a child without first making a geometry request to the widget's own parent (the original requestor's grandparent). If the request to the grandparent would allow the parent to satisfy the original request, the geometry manager can make the intermediate geometry request as if it were the originator. On the other hand, if the geometry manager already has determined that the original request cannot be completely satisfied (for example, if it always denies position changes), it needs to tell the grandparent to respond to the intermediate request without actually changing the geometry because it does not know if the child will accept the compromise. To accomplish this, the geometry manager uses XtCWQueryOnly in the intermediate request.

When XtCWQueryOnly is used, the geometry manager needs to cache enough information to exactly reconstruct the intermediate request. If the grandparent's response to the intermediate query was XtGeometryAlmost, the geometry manager needs to cache the entire reply geometry in the event the child accepts the parent's compromise.

If the grandparent's response was XtGeometryAlmost, it may also be necessary to cache the entire reply geometry from the grandparent when XtCWQueryOnly is not used. If the geometry manager is still able to satisfy the original request, it may immediately accept the grandparent's compromise and then act on the child's request. If the grandparent's compromise geometry is insufficient to allow the child's request and if the geometry manager is willing to offer a different compromise to the child, the grandparent's compromise should not be accepted until the child has accepted the new compromise.

Note that a compromise geometry returned with XtGeometryAlmost is guaranteed only for the next call to the same widget; therefore, a cache of size 1 is sufficient.

6.5. Child Geometry Management: the geometry_manager Procedure

The geometry_manager procedure pointer in a composite widget class is of type XtGeometryHandler.

typedef XtGeometryResult (*XtGeometryHandler)(Widget, XtWidgetGeometry*, XtWidgetGeometry

w
request
geometry_return

Passes the widget making the request.

Passes the new geometry the child desires.

Passes a geometry structure in which the geometry manager may store a compromise.
A class can inherit its superclass's geometry manager during class initialization.

A bit set to zero in the request's `request_mode` field means that the child widget does not care about the value of the corresponding field, so the geometry manager can change this field as it wishes. A bit set to 1 means that the child wants that geometry element changed to the value in the corresponding field.

If the geometry manager can satisfy all changes requested and if `XtCWQueryOnly` is not specified, it updates the widget's `x`, `y`, `width`, `height`, and `border_width` fields appropriately. Then, it returns `XtGeometryYes`, and the values pointed to by the `geometry_return` argument are undefined. The widget's window is moved and resized automatically by `XtMakeGeometryRequest`.

Homogeneous composite widgets often find it convenient to treat the widget making the request the same as any other widget, including reconfiguring it using `XtConfigureWidget` or `XtResizeWidget` as part of its layout process, unless `XtCWQueryOnly` is specified. If it does this, it should return `XtGeometryDone` to inform `XtMakeGeometryRequest` that it does not need to do the configuration itself.

**Note**

To remain compatible with layout techniques used in older widgets (before `XtGeometryDone` was added to the Intrinsics), a geometry manager should avoid using `XtResizeWidget` or `XtConfigureWidget` on the child making the request because the layout process of the child may be in an intermediate state in which it is not prepared to handle a call to its resize procedure. A self-contained widget set may choose this alternative geometry management scheme, however, provided that it clearly warns widget developers of the compatibility consequences.

Although `XtMakeGeometryRequest` resizes the widget's window (if the geometry manager returns `XtGeometryYes`), it does not call the widget class's resize procedure. The requesting widget must perform whatever resizing calculations are needed explicitly.

If the geometry manager disallows the request, the widget cannot change its geometry. The values pointed to by `geometry_return` are undefined, and the geometry manager returns `XtGeometryNo`.

Sometimes the geometry manager cannot satisfy the request exactly but may be able to satisfy a similar request. That is, it could satisfy only a subset of the requests (for example, size but not position) or a lesser request (for example, it cannot make the child as big as the request but it can make the child bigger than its current size). In such cases, the geometry manager fills in the structure pointed to by `geometry_return` with the actual changes it is willing to make, including an appropriate `request_mode` mask, and returns `XtGeometryAlmost`. If a bit in `geometry_return->request_mode` is zero, the geometry manager agrees not to change the corresponding value if `geometry_return` is used immediately in a new request. If a bit is 1, the geometry manager does change that element to the corresponding value in `geometry_return`. More bits may be set in `geometry_return->request_mode` than in the original request if the geometry manager intends to change other fields should the child accept the compromise.

When `XtGeometryAlmost` is returned, the widget must decide if the compromise suggested in `geometry_return` is acceptable. If it is, the widget must not change its geometry directly; rather, it must make another call to `XtMakeGeometryRequest`.

If the next geometry request from this child uses the `geometry_return` values filled in by the geometry manager with an `XtGeometryAlmost` return and if there have been no intervening geometry requests on either its parent or any of its other children, the geometry manager must grant the request, if possible. That is, if the child asks immediately with the returned geometry, it should get an answer of `XtGeometryYes`. However, dynamic behavior in the user's window manager may affect the final outcome.
To return XtGeometryYes, the geometry manager frequently rearranges the position of other managed children by calling XtMoveWidget. However, a few geometry managers may sometimes change the size of other managed children by calling XtResizeWidget or XtConfigureWidget. If XtCWQueryOnly is specified, the geometry manager must return data describing how it would react to this geometry request without actually moving or resizing any widgets.

Geometry managers must not assume that the request and geometry return arguments point to independent storage. The caller is permitted to use the same field for both, and the geometry manager must allocate its own temporary storage, if necessary.

6.6. Widget Placement and Sizing

To move a sibling widget of the child making the geometry request, the parent uses XtMoveWidget.

```c
void XtMoveWidget(w, x, y)
    Widget w;
    Position x;
    Position y;

w Specifies the widget. Must be of class RectObj or any subclass thereof.
x y Specify the new widget x and y coordinates.
```

The XtMoveWidget function returns immediately if the specified geometry fields are the same as the old values. Otherwise, XtMoveWidget writes the new x and y values into the object and, if the object is a widget and is realized, issues an Xlib XMoveWindow call on the widget’s window.

To resize a sibling widget of the child making the geometry request, the parent uses XtResizeWidget.

```c
void XtResizeWidget(w, width, height, border_width)
    Widget w;
    Dimension width;
    Dimension height;
    Dimension border_width;

w Specifies the widget. Must be of class RectObj or any subclass thereof.
width height border_width Specify the new widget size.
```

The XtResizeWidget function returns immediately if the specified geometry fields are the same as the old values. Otherwise, XtResizeWidget writes the new width, height, and border_width values into the object and, if the object is a widget and is realized, issues an XConfigureWindow call on the widget’s window.

If the new width or height is different from the old values, XtResizeWidget calls the object’s resize procedure to notify it of the size change.

To move and resize the sibling widget of the child making the geometry request, the parent uses XtConfigureWidget.
void XtConfigureWidget(w, x, y, width, height, border_width)
    Widget w;
    Position x;
    Position y;
    Dimension width;
    Dimension height;
    Dimension border_width;

w       Specifies the widget. Must be of class RectObj or any subclass thereof.
x
y       Specify the new widget x and y coordinates.
width
height
border_width       Specify the new widget size.

The XtConfigureWidget function returns immediately if the specified new geometry fields are all equal to the current values. Otherwise, XtConfigureWidget writes the new x, y, width, height, and border_width values into the object and, if the object is a widget and is realized, makes an Xlib XConfigureWindow call on the widget's window.

If the new width or height is different from its old value, XtConfigureWidget calls the object's resize procedure to notify it of the size change; otherwise, it simply returns.

To resize a child widget that already has the new values of its width, height, and border width, the parent uses XtResizeWindow.

void XtResizeWindow(w)
    Widget w;

w       Specifies the widget. Must be of class Core or any subclass thereof.

The XtResizeWindow function calls the XConfigureWindow Xlib function to make the window of the specified widget match its width, height, and border width. This request is done unconditionally because there is no inexpensive way to tell if these values match the current values. Note that the widget's resize procedure is not called.

There are very few times to use XtResizeWindow; instead, the parent should use XtResizeWidget.

6.7. Preferred Geometry
Some parents may be willing to adjust their layouts to accommodate the preferred geometries of their children. They can use XtQueryGeometry to obtain the preferred geometry and, as they see fit, can use or ignore any portion of the response.

To query a child widget's preferred geometry, use XtQueryGeometry.

XtGeometryResult XtQueryGeometry(w, intended, preferred_return)
    Widget w;
    XtWidgetGeometry *intended, *preferred_return;

w       Specifies the widget. Must be of class RectObj or any subclass thereof.
intended       Specifies the new geometry the parent plans to give to the child, or NULL.
preferred_return Returns the child widget's preferred geometry.

To discover a child's preferred geometry, the child's parent stores the new geometry in the corresponding fields of the intended structure, sets the corresponding bits in intended.request_mode, and calls XtQueryGeometry. The parent should set only those fields that are important to it so that the child can determine whether it may be able to attempt
changes to other fields.

*XtQueryGeometry* clears all bits in the *preferred_return->request_mode* field and checks the *query_geometry* field of the specified widget's class record. If *query_geometry* is not NULL, *XtQueryGeometry* calls the query_geometry procedure and passes as arguments the specified widget, *intended*, and *preferred_return* structures. If the *intended* argument is NULL, *XtQueryGeometry* replaces it with a pointer to an *XtWidgetGeometry* structure with *request_mode* equal to zero before calling the query_geometry procedure.

Note

If *XtQueryGeometry* is called from within a geometry_manager procedure for the widget that issued *XtMakeGeometryRequest* or *XtMakeResizeRequest*, the results are not guaranteed to be consistent with the requested changes. The change request passed to the geometry manager takes precedence over the preferred geometry.

The query_geometry procedure pointer is of type *XtGeometryHandler*.

```c
typedef XtGeometryResult (*XtGeometryHandler)(Widget, XtWidgetGeometry*, XtWidgetGeometry*);

Widget w;
XtWidgetGeometry *request,
XtWidgetGeometry *preferred_return;
```

`w` Passes the child widget whose preferred geometry is required.

`request` Passes the geometry changes which the parent plans to make.

`preferred_return` Passes a structure in which the child returns its preferred geometry.

The query_geometry procedure is expected to examine the bits set in *request->request_mode*, evaluate the preferred geometry of the widget, and store the result in *preferred_return* (setting the bits in *preferred_return->request_mode* corresponding to those geometry fields that it cares about). If the proposed geometry change is acceptable without modification, the query_geometry procedure should return XtGeometryYes. If at least one field in *preferred_return* with a bit set in *preferred_return->request_mode* is different from the corresponding field in *request* or if a bit was set in *preferred_return->request_mode* that was not set in the request, the query_geometry procedure should return XtGeometryAlmost. If the preferred geometry is identical to the current geometry, the query_geometry procedure should return XtGeometryNo.

Note

The query_geometry procedure may assume that no *XtMakeResizeRequest* or *XtMakeGeometryRequest* is in progress for the specified widget; that is, it is not required to construct a reply consistent with the requested geometry if such a request were actually outstanding.

After calling the query_geometry procedure or if the *query_geometry* field is NULL, *XtQueryGeometry* examines all the unset bits in *preferred_return->request_mode* and sets the corresponding fields in *preferred_return* to the current values from the widget instance. If CWSStackMode is not set, the *stack_mode* field is set to XtSMDontChange. *XtQueryGeometry* returns the value returned by the query_geometry procedure or XtGeometryYes if the *query_geometry* field is NULL.

Therefore, the caller can interpret a return of XtGeometryYes as not needing to evaluate the contents of the reply and, more important, not needing to modify its layout plans. A return of XtGeometryAlmost means either that both the parent and the child expressed interest in at least one common field and the child’s preference does not match the parent’s intentions or that
the child expressed interest in a field that the parent might need to consider. A return value of
XtGeometryNo means that both the parent and the child expressed interest in a field and that
the child suggests that the field’s current value in the widget instance is its preferred value. In
addition, whether or not the caller ignores the return value or the reply mask, it is guaranteed
that the preferred_return structure contains complete geometry information for the child.

Parents are expected to call XtQueryGeometry in their layout routine and wherever else the
information is significant after change_managed has been called. The first time it is invoked,
the changed managed procedure may assume that the child’s current geometry is its preferred
geometry. Thus, the child is still responsible for storing values into its own geometry during
its initialize procedure.

6.8. Size Change Management: the resize Procedure

A child can be resized by its parent at any time. Widgets usually need to know when they
have changed size so that they can lay out their displayed data again to match the new size.
When a parent resizes a child, it calls XtResizeWidget, which updates the geometry fields in
the widget, configures the window if the widget is realized, and calls the child’s resize pro¬
cedure to notify the child. The resize procedure pointer is of type XtWidgetProc.

If a class need not recalculate anything when a widget is resized, it can specify NULL for the
resize field in its class record. This is an unusual case and should occur only for widgets with
very trivial display semantics. The resize procedure takes a widget as its only argument. The
x, y, width, height, and border_width fields of the widget contain the new values. The resize
procedure should recalculate the layout of internal data as needed. (For example, a centered
Label in a window that changes size should recalculate the starting position of the text.) The
widget must obey resize as a command and must not treat it as a request. A widget must not
issue an XtMakeGeometryRequest or XtMakeResizeRequest call from its resize procedure.
Chapter 7
Event Management

While Xlib allows the reading and processing of events anywhere in an application, widgets in the X Toolkit neither directly read events nor grab the server or pointer. Widgets register procedures that are to be called when an event or class of events occurs in that widget.

A typical application consists of startup code followed by an event loop that reads events and dispatches them by calling the procedures that widgets have registered. The default event loop provided by the Intrinsics is XtAppMainLoop.

The event manager is a collection of functions to perform the following tasks:

- Add or remove event sources other than X server events (in particular, timer interrupts and file input).
- Query the status of event sources.
- Add or remove procedures to be called when an event occurs for a particular widget.
- Enable and disable the dispatching of user-initiated events (keyboard and pointer events) for a particular widget.
- Constrain the dispatching of events to a cascade of pop-up widgets.
- Register procedures to be called when specific events arrive.

Most widgets do not need to call any of the event handler functions explicitly. The normal interface to X events is through the higher-level translation manager, which maps sequences of X events, with modifiers, into procedure calls. Applications rarely use any of the event manager routines besides XtAppMainLoop.

7.1. Adding and Deleting Additional Event Sources

While most applications are driven only by X events, some applications need to incorporate other sources of input into the Intrinsics event-handling mechanism. The event manager provides routines to integrate notification of timer events and file data pending into this mechanism.

The next section describes functions that provide input gathering from files. The application registers the files with the Intrinsics read routine. When input is pending on one of the files, the registered callback procedures are invoked.

7.1.1. Adding and Removing Input Sources

To register a new file as an input source for a given application context, use XtAppAddInput.

`XtInputId XtAppAddInput(app_context, source, condition, proc, client_data)`

- `app_context`: Specifies the application context that identifies the application.
- `source`: Specifies the source file descriptor on a POSIX-based system or other operating-system-dependent device specification.
condition Specifies the mask that indicates a read, write, or exception condition or some other operating-system-dependent condition.

proc Specifies the procedure to be called when the condition is found.

client_data Specifies an argument passed to the specified procedure when it is called.

The `XtAppAddInput` function registers with the Intrinsics read routine a new source of events, which is usually file input but can also be file output. Note that `file` should be loosely interpreted to mean any sink or source of data. `XtAppAddInput` also specifies the conditions under which the source can generate events. When an event is pending on this source, the callback procedure is called.

The legal values for the `condition` argument are operating-system-dependent. On a POSIX-based system, `source` is a file number and the condition is some union of the following:

- `XtlnputReadMask` Specifies that `proc` is to be called when `source` has data to be read.
- `XtlnputWriteMask` Specifies that `proc` is to be called when `source` is ready for writing.
- `XtlnputExceptMask` Specifies that `proc` is to be called when `source` has exception data.

Callback procedure pointers used to handle file events are of type `XtlnputCallbackProc`.

```c
typedef void (*XtlnputCallbackProc)(XtPointcr, ini*, Xtlnputld*);
```

- `client_data` Passes the client data argument that was registered for this procedure in `XtAppAddInput`.
- `source` Passes the source file descriptor generating the event.
- `id` Passes the id returned from the corresponding `XtAppAddInput` call.

To discontinue a source of input, use `XtRemoveInput`.

```c
void XtRemoveInput(id)
    Xtlnputld id;
```

- `id` Specifies the id returned from the corresponding `XtAppAddInput` call.

The `XtRemoveInput` function causes the Intrinsics read routine to stop watching for events from the file source specified by `id`.

### 7.1.2. Adding and Removing Timeouts

The timeout facility notifies the application or the widget through a callback procedure that a specified time interval has elapsed. Timeout values are uniquely identified by an interval id.

To register a timeout callback, use `XtAppAddTimeOut`.

```c
XtIntervalld XtAppAddTimeOut(app_context, interval, proc, client_data)
    XtAppContext app_context;
    unsigned long interval;
    XtTimerCallbackProc proc;
    XtPointer client_data;
```

- `app_context` Specifies the application context for which the timer is to be set.
- `interval` Specifies the time interval in milliseconds.
- `proc` Specifies the procedure to be called when the time expires.
- `client_data` Specifies an argument passed to the specified procedure when it is called.

85
The **XtAppAddTimeOut** function creates a timeout and returns an identifier for it. The timeout value is set to *interval*. The callback procedure *proc* is called when **XtAppNextEvent** or **XtAppProcessEvent** is next called after the time interval elapses, and then the timeout is removed.

Callback procedure pointers used with timeouts are of type **XtTimerCallbackProc**.

```c
typedef void (*XtTimerCallbackProc)(XtPointer, XtIntervalId*);
```

- `client_data` Passes the client data argument that was registered for this procedure in **XtAppAddTimeOut**.
- `timer` Passes the id returned from the corresponding **XtAppAddTimeOut** call.

To clear a timeout value, use **XtRemoveTimeOut**.

```c
void XtRemoveTimeOut(timer);
```

- `timer` Specifies the id for the timeout request to be cleared.

The **XtRemoveTimeOut** function removes the pending timeout. Note that timeouts are automatically removed once they trigger.

### 7.2. Constraining Events to a Cascade of Widgets

Modal widgets are widgets that, except for the input directed to them, lock out user input to the application.

When a modal menu or modal dialog box is popped up using **XtPopup**, user events (keyboard and pointer events) that occur outside the modal widget should be delivered to the modal widget or ignored. In no case will user events be delivered to a widget outside the modal widget. Menus can pop up submenus, and dialog boxes can pop up further dialog boxes, to create a pop-up cascade. In this case, user events may be delivered to one of several modal widgets in the cascade.

Display-related events should be delivered outside the modal cascade so that exposure events and the like keep the application's display up-to-date. Any event that occurs within the cascade is delivered as usual. The user events delivered to the most recent spring-loaded shell in the cascade when they occur outside the cascade are called remap events and are **KeyPress**, **KeyRelease**, **ButtonPress**, and **ButtonRelease**. The user events ignored when they occur outside the cascade are **MotionNotify** and **EnterNotify**. All other events are delivered normally. In particular, note that this is one way in which widgets can receive **LeaveNotify** events without first receiving **EnterNotify** events; they should be prepared to deal with this, typically by ignoring any unmatched **LeaveNotify** events.

**XtPopup** uses the **XtAddGrab** and **XtRemoveGrab** functions to constrain user events to a modal cascade and subsequently to remove a grab when the modal widget is popped down.

To constrain or redirect user input to a modal widget, use **XtAddGrab**.

```c
void XtAddGrab(w, exclusive, spring_loaded);
```

- `w` Specifies the widget to add to the modal cascade. Must be of class Core or any subclass thereof.
**exclusive** Specifies whether user events should be dispatched exclusively to this widget or also to previous widgets in the cascade.

**spring_loaded** Specifies whether this widget was popped up because the user pressed a pointer button.

The `XtAddGrab` function appends the widget to the modal cascade and checks that `exclusive` is **True** if `spring_loaded` is **True**. If this condition is not met, `XtAddGrab` generates a warning message.

The modal cascade is used by `XtDispatchEvent` when it tries to dispatch a user event. When at least one modal widget is in the widget cascade, `XtDispatchEvent` first determines if the event should be delivered. It starts at the most recent cascade entry and follows the cascade up to and including the most recent cascade entry added with the `exclusive` parameter **True**.

This subset of the modal cascade along with all descendants of these widgets comprise the active subset. User events that occur outside the widgets in this subset are ignored or remapped. Modal menus with submenus generally add a submenu widget to the cascade with `exclusive` **False**. Modal dialog boxes that need to restrict user input to the most deeply nested dialog box add a subdialog widget to the cascade with `exclusive` **True**. User events that occur within the active subset are delivered to the appropriate widget, which is usually a child or further descendant of the modal widget.

Regardless of where in the application they occur, remap events are always delivered to the most recent widget in the active subset of the cascade registered with `spring_loaded` **True**, if any such widget exists. If the event occurred in the active subset of the cascade but outside the spring-loaded widget, it is delivered normally before being delivered also to the spring-loaded widget. Regardless of where it is dispatched, the Intrinsics do not modify the contents of the event.

To remove the redirection of user input to a modal widget, use `XtRemoveGrab`.

```c
void XtRemoveGrab(wid)
    Widget wid;
```

`w` Specifies the widget to remove from the modal cascade.

The `XtRemoveGrab` function removes widgets from the modal cascade starting at the most recent widget up to and including the specified widget. It issues a warning if the specified widget is not on the modal cascade.

### 7.2.1. Requesting Key and Button Grabs

The Intrinsics provide a set of key and button grab interfaces that are parallel to those provided by `Xlib` and that allow the Intrinsics to modify event dispatching when necessary. `X Toolkit` applications and widgets that need to passively grab keys or buttons or actively grab the keyboard or pointer should use the following Intrinsics routines rather than the corresponding `Xlib` routines.

To passively grab a single key of the keyboard, use `XtGrabKey`.

```c
void XtGrabKey(widget, keycode, modifiers, owner_events, pointer_mode, keyboard_mode)
    Widget widget;
    KeyCode keycode;
    Modifiers modifiers;
    Boolean owner_events;
    int pointer_mode, keyboard_mode;
```

`widget` Specifies the widget in whose window the key is to be grabbed. Must be of class `Core` or any subclass thereof.
keycode
modifiers
owner_events
pointer_mode
keyboard_mode
Specify arguments to XGrabKey; see Section 12.2 in Xlib – C Language X Interface.

XtGrabKey calls XGrabKey specifying the widget’s window as the grab window if the widget is realized. The remaining arguments are exactly as for XGrabKey. If the widget is not realized, or is later unrealized, the call to XGrabKey will be performed (again) when the widget is realized and its window becomes mapped. In the future, if XtDispatchEvent is called with a.KeyPress event matching the specified keycode and modifiers (which may be AnyKey or AnyModifier, respectively) for the widget’s window, the Intrinsics will call XtUngrabKeyboard with the timestamp from the KeyPress event if either of the following conditions is true:
- There is a modal cascade and the widget is not in the active subset of the cascade and the keyboard was not previously grabbed, or
- XFilterEvent returns True.

To cancel a passive key grab, use XtUngrabKey.

```c
void XtUngrabKey(Widget widget, KeyCode keycode, Modifiers modifiers);
```

widget Specifies the widget in whose window the key was grabbed.
keycode Specify arguments to XUngrabKey; see Section 12.2 in Xlib – C Language X Interface.
modifiers

The XtUngrabKey procedure calls XUngrabKey specifying the widget’s window as the ungrab window if the widget is realized. The remaining arguments are exactly as for XUngrabKey. If the widget is not realized, XtUngrabKey removes a deferred XtGrabKey request, if any, for the specified widget, keycode, and modifiers.

To actively grab the keyboard, use XtGrabKeyboard.

```c
int XtGrabKeyboard(Widget widget, owner_events, pointer_mode, keyboard_mode, time);
```

widget Specifies the widget for whose window the keyboard is to be grabbed. Must be of class Core or any subclass thereof.
owner_events
pointer_mode
keyboard_mode
time  Specify arguments to XGrabKeyboard; see Section 12.2 in Xlib – C Language X Interface.

If the specified widget is realized XtGrabKeyboard calls XGrabKeyboard specifying the widget’s window as the grab window. The remaining arguments and return value are exactly
as for `XGrabKeyboard`. If the widget is not realized, `XGrabKeyboard` immediately returns `GrabNotViewable`. No future automatic ungrab is implied by `XtGrabKeyboard`.

To cancel an active keyboard grab, use `XtUngrabKeyboard`.

```c
void XtUngrabKeyboard(widget, time) 
    Widget widget;
    Time time;

    widget Specifies the widget that has the active keyboard grab.
    time Specifies the additional argument to `XUngrabKeyboard`; see Section 12.2 in Xlib – C Language X Interface.
```

`XtUngrabKeyboard` calls `XUngrabKeyboard` with the specified time.

To passively grab a single pointer button, use `XtGrabButton`.

```c
void XtGrabButton(widget, button, modifiers, owner_events, event_mask, pointer_mode, keyboard_mode, confine_to, cursor) 
    Widget widget;
    int button;
    Modifiers modifiers;
    Boolean owner_events;
    unsigned int event_mask;
    int pointer_mode, keyboard_mode;
    Window confine_to;
    Cursor cursor;

    widget Specifies the widget in whose window the button is to be grabbed. Must be of class Core or any subclass thereof.
    button
    modifiers
    owner_events
    event_mask
    pointer_mode
    keyboard_mode
    confine_to
    cursor Specify arguments to `XGrabButton`; see Section 12.1 in Xlib – C Language X Interface.
```

`XtGrabButton` calls `XGrabButton` specifying the widget’s window as the grab window if the widget is realized. The remaining arguments are exactly as for `XGrabButton`. If the widget is not realized, or is later unrealized, the call to `XGrabButton` will be performed (again) when the widget is realized and its window becomes mapped. In the future, if `XtDispatchEvent` is called with a `ButtonPress` event matching the specified button and modifiers (which may be `AnyButton` or `AnyModifier`, respectively) for the widget’s window, the Intrinsics will call `XtUngrabPointer` with the timestamp from the `ButtonPress` event if either of the following conditions is true:

- There is a modal cascade and the widget is not in the active subset of the cascade and the pointer was not previously grabbed, or
- `XFilterEvent` returns `True`.

To cancel a passive button grab, use `XtUngrabButton`. 

89
void XtUngrabButton(widget, button, modifiers)
   Widget widget;
   unsigned int button;
   Modifiers modifiers;

   widget Specifies the widget in whose window the button was grabbed.
   button modifiers Specify arguments to XUngrabButton; see Section 12.1 in Xlib – C Language X Interface.

   The XtUngrabButton procedure calls XUngrabButton specifying the widget's window as the ungrab window if the widget is realized. The remaining arguments are exactly as for XUngrabButton. If the widget is not realized, XtUngrabButton removes a deferred XtGrabButton request, if any, for the specified widget, button, and modifiers.

To actively grab the pointer, use XtGrabPointer.

int XtGrabPointer(widget, owner_events, event_mask, pointer_mode, keyboard_mode, confine_to, cursor, time)
   Widget widget;
   Boolean owner_events;
   unsigned int event_mask;
   int pointer_mode, keyboard_mode;
   Window confine_to;
   Cursor cursor;
   Time time;

   widget Specifies the widget for whose window the pointer is to be grabbed. Must be of class Core or any subclass thereof.

   owner_events event_mask pointer_mode keyboard_mode confine_to cursor
time Specify arguments to XGrabPointer; see Section 12.1 in Xlib – C Language X Interface.

   If the specified widget is realized, XtGrabPointer calls XGrabPointer, specifying the widget's window as the grab window. The remaining arguments and return value are exactly as for XGrabPointer. If the widget is not realized, XGrabPointer immediately returns GrabNotViewable. No future automatic ungrab is implied by XtGrabPointer.

To cancel an active pointer grab, use XtUngrabPointer.

void XtUngrabPointer(widget, time)
   Widget widget;
   Time time;

   widget Specifies the widget that has the active pointer grab.
   time Specifies the time argument to XUngrabPointer; see Section 12.1 in Xlib – C Language X Interface.

   XtUngrabPointer calls XUngrabPointer with the specified time.
7.3. Focusing Events on a Child

To redirect keyboard input to a normal descendant of a widget without calling XSetInputFocus, use XtSetKeyboardFocus.

```c
void XtSetKeyboardFocus(subtree, descendant)
    Widget subtree, descendant;
```

*subtree* Specifies the subtree of the hierarchy for which the keyboard focus is to be set. Must be of class Core or any subclass thereof.

*descendant* Specifies either the normal (non-pop-up) descendant of *subtree* to which keyboard events are logically directed, or *None*. It is not an error to specify *None* when no input focus was previously set. Must be of class Object or any subclass thereof.

XtSetKeyboardFocus causes XtDispatchEvent to remap keyboard events occurring within the specified subtree and dispatch them to the specified descendant widget or to an ancestor. If the descendant's class is not a subclass of Core, the descendant is replaced by its closest windowed ancestor.

When there is no modal cascade, keyboard events can be dispatched to a widget in one of five ways. Assume the server delivered the event to the window for widget E (because of X input focus, key or keyboard grabs, or pointer position).

- If neither E nor any of E's ancestors have redirected the keyboard focus, or if the event activated a grab for E as specified by a call to XtGrabKey with any value of *owner_events*, or if the keyboard is actively grabbed by E with *owner_events* *False* via XtGrabKeyboard or XtGrabKey on a previous key press, the event is dispatched to E.
- Beginning with the ancestor of E closest to the root that has redirected the keyboard focus or E if no such ancestor exists, if the target of that focus redirection has in turn redirected the keyboard focus, recursively follow this focus chain to find a widget F that has not redirected focus.
  - If E is the final focus target widget F or a descendant of F, the event is dispatched to E.
  - If E is not F, an ancestor of F, or a descendant of F, and the event activated a grab for E as specified by a call to XtGrabKey for E, XtUngrabKeyboard is called.
  - If E is an ancestor of F, and the event is a key press, and either
    + E has grabbed the key with XtGrabKey and *owner_events* *False*, or
    + E has grabbed the key with XtGrabKey and *owner_events* *True*, and the coordinates of the event are outside the rectangle specified by E's geometry, then the event is dispatched to E.
  - Otherwise, define A as the closest common ancestor of E and F:
    + If there is an active keyboard grab for any widget via either XtGrabKeyboard or XtGrabKey on a previous key press, or if no widget between F and A (non-inclusive) has grabbed the key and modifier combination with XtGrabKey and any value of *owner_events*, the event is dispatched to F.
    + Else, the event is dispatched to the ancestor of F closest to A that has grabbed the key and modifier combination with XtGrabKey.

When there is a modal cascade, if the final destination widget as identified above is in the active subset of the cascade, the event is dispatched; otherwise the event is remapped to a spring-loaded shell or discarded. Regardless of where it is dispatched, the Intrinsics do not modify the contents of the event.

When *subtree* or one of its descendants acquires the X input focus or the pointer moves into the subtree such that keyboard events would now be delivered to the subtree, a FocusIn event is generated for the descendant if FocusChange events have been selected by the descendant. Similarly, when *subtree* loses the X input focus or the keyboard focus for one of its ancestors,
a **FocusOut** event is generated for descendant if **FocusChange** events have been selected by the descendant.

A widget tree may also actively manage the X server input focus. To do so, a widget class specifies an accept_focus procedure.

The accept_focus procedure pointer is of type **XtAcceptFocusProc**.

```c
typedef Boolean (*XtAcceptFocusProc)(Widget, Time*);
```

```c
Widget w;
Time *time;
```

- **w** Specifies the widget.
- **time** Specifies the X time of the event causing the accept focus.

Widgets that need the input focus can call **XSetInputFocus** explicitly, pursuant to the restrictions of the *Inter-Client Communication Conventions Manual*. To allow outside agents, such as the parent, to cause a widget to take the input focus, every widget exports an accept_focus procedure. The widget returns a value indicating whether it actually took the focus or not, so that the parent can give the focus to another widget. Widgets that need to know when they lose the input focus must use the Xlib focus notification mechanism explicitly (typically by specifying translations for **FocusIn** and **FocusOut** events). Widgets classes that never want the input focus should set the **accept_focus** field to NULL.

To call a widget's accept_focus procedure, use **XtCallAcceptFocus**.

```c
Boolean XtCallAcceptFocus(w, time)
    Widget w;
    Time *time;
```

- **w** Specifies the widget. Must be of class Core or any subclass thereof.
- **time** Specifies the X time of the event that is causing the focus change.

The **XtCallAcceptFocus** function calls the specified widget's accept_focus procedure, passing it the specified widget and time, and returns what the accept_focus procedure returns. If **accept_focus** is NULL, **XtCallAcceptFocus** returns False.

### 7.4. Querying Event Sources

The event manager provides several functions to examine and read events (including file and timer events) that are in the queue. The next three functions are Intrinsics equivalents of the **XPending**, **XPeekEvent**, and **XNextEvent** Xlib calls.

To determine if there are any events on the input queue for a given application, use **XtAppPending**.

```c
XtInputMask XtAppPending(app_context)
    XtAppContext app_context;
```

- **app_context** Specifies the application context that identifies the application to check.

The **XtAppPending** function returns a nonzero value if there are events pending from the X server, timer pending, or other input sources pending. The value returned is a bit mask that is the OR of **XtIMXEvent**, **XtIMTimer**, and **XtIMAlternateInput** (see **XtAppProcessEvent**). If there are no events pending, **XtAppPending** flushes the output buffers of each Display in the application context and returns zero.

To return the event from the head of a given application's input queue without removing input from the queue, use **XtAppPeekEvent**.
X Toolkit Intrinsics

Boolean XtAppPeekEvent(app_context, event_return)
    XtApplicationContext app_context,
    XEvent *event_return;

app_context Specifies the application context that identifies the application.

event_return Returns the event information to the specified event structure.

If there is an X event in the queue, XtAppPeekEvent copies it into event_return and returns True. If no X input is on the queue, XtAppPeekEvent flushes the output buffers of each Display in the application context and blocks until some input is available (possibly calling some timeout callbacks in the interim). If the next available input is an X event, XtAppPeekEvent fills in event_return and returns True. Otherwise, the input is for an input source registered with XtAppAddInput, and XtAppPeekEvent returns False.

To remove and return the event from the head of a given application’s X event queue, use XtAppNextEvent.

void XtAppNextEvent(app_context, event_return)
    XtApplicationContext app_context,
    XEvent *event_return;

app_context Specifies the application context that identifies the application.

event_return Specifies the event information to the specified event structure.

If the X event queue is empty, XtAppNextEvent flushes the X output buffers of each Display in the application context and waits for an X event while looking at the other input sources and timeout values and calling any callback procedures triggered by them. This wait time can be used for background processing; see Section 7.8.

7.5. Dispatching Events

The Intrinsics provide functions that dispatch events to widgets or other application code. Every client interested in X events on a widget uses XtAddEventHandler to register which events it is interested in and a procedure (event handler) to be called when the event happens in that window. The translation manager automatically registers event handlers for widgets that use translation tables; see Chapter 10.

Applications that need direct control of the processing of different types of input should use XtAppProcessEvent.

void XtAppProcessEvent(app_context, mask)
    XtApplicationContext app_context,
    XtInputMask mask;

app_context Specifies the application context that identifies the application for which to process input.

mask Specifies what types of events to process. The mask is the bitwise inclusive OR of any combination of XtIMEvent, XtIMTimer, and XtIMAternaInput. As a convenience, Intrinsic.h defines the symbolic name XtIMAll to be the bitwise inclusive OR of these three event types.

The XtAppProcessEvent function processes one timer, input source, or X event. If there is no event or input of the appropriate type to process, then XtAppProcessEvent blocks until there is. If there is more than one type of input available to process, it is undefined which will get processed. Usually, this procedure is not called by client applications; see XtAppMainLoop. XtAppProcessEvent processes timer events by calling any appropriate timer callbacks, input sources by calling any appropriate input callbacks, and X events by calling XtDispatchEvent.
When an X event is received, it is passed to \texttt{XtDispatchEvent}, which calls the appropriate event handlers and passes them the widget, the event, and client-specific data registered with each procedure. If no handlers for that event are registered, the event is ignored and the dispatcher simply returns.

To dispatch an event returned by \texttt{XtAppNextEvent}, retrieved directly from the Xlib queue, or synthetically constructed, to any registered event filters or event handlers call \texttt{XtDispatchEvent}.

\begin{verbatim}
Boolean XtDispatchEvent(event)
    XEvent *event;

    event Specifies a pointer to the event structure to be dispatched to the appropriate event handlers.

    The XtDispatchEvent function first calls XFilterEvent with the event and the window of the widget to which the Intrinsics intend to dispatch the event, or the event window if the Intrinsics would not dispatch the event to any handlers. If XFilterEvent returns True and the event activated a server grab as identified by a previous call to XtGrabKey or XtGrabButton, XtDispatchEvent calls XtUngrabKeyboard or XtUngrabPointer with the timestamp from the event and immediately returns True. If XFilterEvent returns True and a grab was not activated, XtDispatchEvent just immediately returns True. Otherwise, XtDispatchEvent sends the event to the event handler functions that have been previously registered with the dispatch routine. XtDispatchEvent returns True if XFilterEvent returned True, or if the event was dispatched to some handler and False if it found no handler to which to dispatch the event. XtDispatchEvent records the last timestamp in any event that contains a timestamp (see XtLastTimestampProcessed), regardless of whether it was filtered or dispatched. If a modal cascade is active with spring\_loaded True, and if the event is a remap event as defined by XtAddGrab, XtDispatchEvent may dispatch the event a second time. If it does so, XtDispatchEvent will call XFilterEvent again with the window of the spring-loaded widget prior to the second dispatch and if XFilterEvent returns True, the second dispatch will not be performed.
\end{verbatim}

7.6. The Application Input Loop

To process all input from a given application in a continuous loop, use the convenience procedure \texttt{XtAppMainLoop}.

\begin{verbatim}
void XtAppMainLoop(app_context)
    XtAppContext app_context;

    app_context Specifies the application context that identifies the application.

    The XtAppMainLoop function first reads the next incoming X event by calling XtAppNextEvent and then dispatches the event to the appropriate registered procedure by calling XtDispatchEvent. This constitutes the main loop of X Toolkit applications, and, as such, it does not return. Applications are expected to exit in response to some user action within a callback procedure. There is nothing special about XtAppMainLoop; it is simply an infinite loop that calls XtAppNextEvent and then XtDispatchEvent.

    Applications can provide their own version of this loop, which tests some global termination flag or tests that the number of top-level widgets is larger than zero before circling back to the call to XtAppNextEvent.
\end{verbatim}
7.7. Setting and Checking the Sensitivity State of a Widget

Many widgets have a mode in which they assume a different appearance (for example, are grayed out or stippled), do not respond to user events, and become dormant.

When dormant, a widget is considered to be insensitive. If a widget is insensitive, the event manager does not dispatch any events to the widget with an event type of KeyPress, KeyRelease, ButtonPress, ButtonRelease, MotionNotify, EnterNotify, LeaveNotify, FocusIn, or FocusOut.

A widget can be insensitive because its sensitive field is False or because one of its ancestors is insensitive and thus the widget's ancestor_sensitive field also is False. A widget can but does not need to distinguish these two cases visually.

Note

Pop-up shells will have ancestor_sensitive False if the parent was insensitive when the shell was created. Since XtSetSensitive on the parent will not modify the resource of the pop-up child, clients are advised to include a resource specification of the form "*TransientShell.ancestorSensitive: True" in the application defaults resource file or to otherwise ensure that the parent is sensitive when creating pop-up shells.

To set the sensitivity state of a widget, use XtSetSensitive.

void XtSetSensitive(w, sensitive)
    Widget w;
    Boolean sensitive;

w Specifies the widget. Must be of class RectObj or any subclass thereof.
sensitive Specifies whether the widget should receive keyboard, pointer, and focus events.

The XtSetSensitive function first calls XtSetValues on the current widget with an argument list specifying the XtNsensitive resource and the new value. If sensitive is False and the widget's class is a subclass of Composite, XtSetSensitive recursively propagates the new value down the child tree by calling XtSetValues on each child to set ancestor_sensitive to False. If sensitive is True and the widget's class is a subclass of Composite and the widget's ancestor_sensitive field is True, XtSetSensitive sets the ancestor_sensitive of each child to True and then recursively calls XtSetValues on each normal descendant that is now sensitive to set ancestor_sensitive to True.

XtSetSensitive calls XtSetValues to change the sensitive and ancestor_sensitive fields of each affected widget. Therefore, when one of these changes, the widget's set_values procedure should take whatever display actions are needed (for example, graying out or stippling the widget).

XtSetSensitive maintains the invariant that if the parent has either sensitive or ancestor_sensitive False, then all children have ancestor_sensitive False.

To check the current sensitivity state of a widget, use XtIsSensitive.

Boolean XtIsSensitive(w)
    Widget w;

w Specifies the object. Must be of class Object or any subclass thereof.

The XtIsSensitive function returns True or False to indicate whether user input events are being dispatched. If object's class is a subclass of RectObj and both sensitive and ancestor_sensitive are True, XtIsSensitive returns True; otherwise, it returns False.
7.8. Adding Background Work Procedures
The Intrinsics have some limited support for background processing. Because most applications spend most of their time waiting for input, you can register an idle-time work procedure that will be called when the toolkit would otherwise block in XtAppNextEvent or XtAppProcessEvent. Work procedure pointers are of type XtWorkProc.

typedef Boolean (*XtWorkProc)(XtPointer);

XtPointer client_data;

client_data Passes the client data specified when the work procedure was registered.

This procedure should return True when it is done to indicate that it should be removed. If the procedure returns False, it will remain registered and will be called again when the application is next idle. Work procedures should be very judicious about how much they do. If they run for more than a small part of a second, interactive feel is likely to suffer.

To register a work procedure for a given application, use XtAppAddWorkProc.

XtWorkProcId XtAppAddWorkProc(app_context, proc, client_data)

XtAppContext app_context;
XtWorkProc proc;
XtPointer client_data;

app_context Specifies the application context that identifies the application.
proc Specifies the procedure to be called when the application is idle.
client_data Specifies the argument passed to the specified procedure when it is called.

The XtAppAddWorkProc function adds the specified work procedure for the application identified by app_context and returns an opaque unique identifier for this work procedure. Multiple work procedures can be registered, and the most recently added one is always the one that is called. However, if a work procedure adds another work procedure, the newly added one has lower priority than the current one.

To remove a work procedure, either return True from the procedure when it is called or use XtRemoveWorkProc.

void XtRemoveWorkProc(id)

XtWorkProcId id;

id Specifies which work procedure to remove.

The XtRemoveWorkProc function explicitly removes the specified background work procedure.

7.9. X Event Filters
The event manager provides filters that can be applied to specific X events. The filters, which screen out events that are redundant or are temporarily unwanted, handle pointer motion compression, enter/leave compression, and exposure compression.

7.9.1. Pointer Motion Compression
Widgets can have a hard time keeping up with a rapid stream of pointer motion events. Further, they usually do not care about every motion event. To throw out redundant motion events, the widget class field compress_motion should be True. When a request for an event would return a motion event, the Intrinsics check if there are any other motion events for the same widget immediately following the current one and, if so, skip all but the last of them.
7.9.2. Enter/Leave Compression
To throw out pairs of enter and leave events that have no intervening events, as can happen when the user moves the pointer across a widget without stopping in it, the widget class field `compress_enterleave` should be True. These enter and leave events are not delivered to the client if they are found together in the input queue.

7.9.3. Exposure Compression
Many widgets prefer to process a series of exposure events as a single expose region rather than as individual rectangles. Widgets with complex displays might use the expose region as a clip list in a graphics context, and widgets with simple displays might ignore the region entirely and redisplay their whole window or might get the bounding box from the region and redisplay only that rectangle.

In either case, these widgets do not care about getting partial exposure events. The `compress_exposure` field in the widget class structure specifies the type and number of exposure events that will be dispatched to the widget’s expose procedure. This field must be initialized to one of the following values:

```c
#define XtExposeNoCompress     ((XtEnum)False)
#define XtExposeCompressSeries  ((XtEnum)True)
#define XtExposeCompressMultiple <implementation-defined>
#define XtExposeCompressMaximal <implementation-defined>
```

optionally ORed with any combination of the following flags (all with implementation-defined values):

- `XtExposeGraphicsExpose`
- `XtExposeGraphicsExposeMerged`
- `XtExposeNoExpose`

If the `compress_exposure` field in the widget class structure does not specify `XtExposeNoCompress`, the event manager calls the widget’s expose procedure only once for a series of exposure events. In this case, all `Exposure` or `GraphicsExposure` events are accumulated into a region. When the final event is received, the event manager replaces the rectangle in the event with the bounding box for the region and calls the widget’s expose procedure, passing the modified exposure event and the region. For more information on regions, see Section 16.5 in `Xlib - C Language X Interface`.

The values have the following interpretation:

- **XtExposeNoCompress**
  No exposure compression is performed; every selected event is individually dispatched to the expose procedure with a `region` argument of NULL.

- **XtExposeCompressSeries**
  Each series of exposure events is coalesced into a single event, which is dispatched when an exposure event with count equal to zero is reached.

- **XtExposeCompressMultiple**
  Consecutive series of exposure events are coalesced into a single event, which is dispatched when an exposure event with count equal to zero is reached and either the event queue is empty or the next event is not an exposure event for the same widget.
X Toolkit Intrinsics

XtExposeCompressMaximal

All expose series currently in the queue for the widget are coalesced into a single event without regard to intervening nonexposure events. If a partial series is in the end of the queue, the Intrinsics will block until the end of the series is received.

The additional flags have the following meaning:

XtExposeGraphicsExpose

Specifies that GraphicsExpose events are also to be dispatched to the expose procedure. GraphicsExpose events will be compressed, if specified, in the same manner as Expose events.

XtExposeGraphicsExposeMerged

Specifies in the case of XtExposeCompressMultiple and XtExposeCompressMaximal that series of GraphicsExpose and Expose events are to be compressed together, with the final event type determining the type of the event passed to the expose procedure. If this flag is not set, then only series of the same event type as the event at the head of the queue are coalesced. This flag also implies XtExposeGraphicsExpose.

XtExposeNoExpose

Specifies that NoExpose events are also to be dispatched to the expose procedure. NoExpose events are never coalesced with other exposure events or with each other.

7.10. Widget Exposure and Visibility

Every primitive widget and some composite widgets display data on the screen by means of direct Xlib calls. Widgets cannot simply write to the screen and forget what they have done. They must keep enough state to redisplay the window or parts of it if a portion is obscured and then reexposed.

7.10.1. Redisplay of a Widget: the expose Procedure

The expose procedure pointer in a widget class is of type XtExposeProc.

typedef void (*XtExposeProc)(Widget, XEvent*, Region);

Widget w;
XEvent *event;
Region region;

w Specifies the widget instance requiring redisplay.

event Specifies the exposure event giving the rectangle requiring redisplay.

region Specifies the union of all rectangles in this exposure sequence.

The redisplay of a widget upon exposure is the responsibility of the expose procedure in the widget’s class record. If a widget has no display semantics, it can specify NULL for the expose field. Many composite widgets serve only as containers for their children and have no
X Toolkit Intrinsics

expose procedure.

Note

If the expose procedure is NULL, XtRealizeWidget fills in a default bit gravity of NorthWestGravity before it calls the widget’s realize procedure.

If the widget’s compress_exposure class field specifies XtExposeNoCompress or the event type is NoExpose (see Section 7.9.3), region is NULL; otherwise, the event is the final event in the compressed series but x, y, width, and height contain the bounding box for region. The region is created and destroyed by the Intrinsics, but the widget is permitted to modify the region contents.

A small simple widget (for example, Label) can ignore the bounding box information in the event and redisplay the entire window. A more complicated widget (for example, Text) can use the bounding box information to minimize the amount of calculation and redisplay it does. A very complex widget uses the region as a clip list in a GC and ignores the event information. The expose procedure is not chained and is therefore responsible for exposure of all superclass data as well as its own.

However, it often is possible to anticipate the display needs of several levels of subclassing. For example, rather than implement separate display procedures for the widgets Label, Pushbutton, and Toggle, you could write a single display routine in Label that uses display state fields like

Boolean invert;
Boolean highlight;
Dimension highlight_width;

Label would have invert and highlight always False and highlight_width zero. Pushbutton would dynamically set highlight and highlight_width, but it would leave invert always False. Finally, Toggle would dynamically set all three. In this case, the expose procedures for Pushbutton and Toggle inherit their superclass’s expose procedure; see Section 1.6.10.

7.10.2. Widget Visibility

Some widgets may use substantial computing resources to produce the data they will display. However, this effort is wasted if the widget is not actually visible on the screen, that is, if the widget is obscured by another application or is iconified.

The visible field in the core widget structure provides a hint to the widget that it need not compute display data. This field is guaranteed to be True by the time an exposure event is processed if any part of the widget is visible but is False if the widget is fully obscured.

Widgets can use or ignore the visible hint. If they ignore it, they should have visible_interest in their widget class record set False. In such cases, the visible field is initialized True and never changes. If visible_interest is True, the event manager asks for VisibilityNotify events for the widget and sets visible to True on VisibilityUnobscured or VisibilityPartiallyObscured events and False on VisibilityFullyObscured events.

7.11. X Event Handlers

Event handlers are procedures called when specified events occur in a widget. Most widgets need not use event handlers explicitly. Instead, they use the Intrinsics translation manager. Event handler procedure pointers are of the type XtEventHandler.
typedef void (*XtEventHandler)(Widget, XtPointer, XEvent*, Boolean*);

    Widget w;
    XtPointer client_data;
    XEvent *event;
    Boolean *continue_to_dispatch;

    w Specifies the widget for which the event arrived.
    client_data Specifies any client-specific information registered with the event handler.
    event Specifies the triggering event.
    continue_to_dispatch Specifies whether the remaining event handlers registered for the current event should be called.

    After receiving an event and before calling any event handlers, the Boolean pointed to by continue_to_dispatch is initialized to True. When an event handler is called, it may decide that further processing of the event is not desirable and may store False in this Boolean, in which case any handlers remaining to be called for the event will be ignored.

    The circumstances under which the Intrinsics may add event handlers to a widget are currently implementation-dependent. Clients must therefore be aware that storing False into the continue_to_dispatch argument can lead to portability problems.

7.11.1. Event Handlers that Select Events

    To register an event handler procedure with the dispatch mechanism, use XtAddEventHandler.

    void XtAddEventHandler(w, event_mask, nonmaskable, proc, client_data)
    
        Widget w;
        EventMask event_mask;
        Boolean nonmaskable;
        XtEventHandler proc;
        XtPointer client_data;

        w Specifies the widget for which this event handler is being registered. Must be of class Core or any subclass thereof.
        event_mask Specifies the event mask for which to call this procedure.
        nonmaskable Specifies whether this procedure should be called on the nonmaskable events (GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage, and MappingNotify).
        proc Specifies the procedure to be called.
        client_data Specifies additional data to be passed to the event handler.

    The XtAddEventHandler function registers a procedure with the dispatch mechanism that is to be called when an event that matches the mask occurs on the specified widget. Each widget has a single registered event handler list, which will contain any procedure–client_data pair exactly once regardless of the manner in which it is registered. If the procedure is already registered with the same client_data value, the specified mask augments the existing mask. If the widget is realized, XtAddEventHandler calls XSelectInput, if necessary. The order in which this procedure is called relative to other handlers registered for the same event is not defined.

    To remove a previously registered event handler, use XtRemoveEventHandler.
void XtRemoveEventHandler(w, event_mask, nonmaskable, proc, client_data)

  Widget w;
  EventMask event_mask;
  Boolean nonmaskable;
  XtEventHandler proc;
  XtPointer client_data;

  w Specifies the widget for which this procedure is registered. Must be of class Core or any subclass thereof.
  event_mask Specifies the event mask for which to unregister this procedure.
  nonmaskable Specifies whether this procedure should be removed on the nonmaskable events (GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage, and MappingNotify).
  proc Specifies the procedure to be removed.
  client_data Specifies the registered client data.

The XtRemoveEventHandler function unregisters an event handler registered with XtAddEventHandler or XtInsertEventHandler for the specified events. The request is ignored if client_data does not match the value given when the handler was registered. If the widget is realized and no other event handler requires the event, XtRemoveEventHandler calls XSelectInput. If the specified procedure has not been registered or if it has been registered with a different value of client_data, XtRemoveEventHandler returns without reporting an error.

To stop a procedure registered with XtAddEventHandler or XtInsertEventHandler from receiving all selected events, call XtRemoveEventHandler with an event_mask of XtAllEvents and nonmaskable True. The procedure will continue to receive any events that have been specified in calls to XtAddRawEventHandler or XtInsertRawEventHandler.

To register an event handler procedure that receives events before or after all previously registered event handlers, use XtInsertEventHandler.

typedef enum {XtListHead, XtListTail} XtListPosition;

void XtInsertEventHandler(w, event_mask, nonmaskable, proc, client_data, position)

  Widget w;
  EventMask event_mask;
  Boolean nonmaskable;
  XtEventHandler proc;
  XtPointer client_data;
  XtListPosition position;

  w Specifies the widget for which this event handler is being registered. Must be of class Core or any subclass thereof.
  event_mask Specifies the event mask for which to call this procedure.
  nonmaskable Specifies whether this procedure should be called on the nonmaskable events (GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage, and MappingNotify).
  proc Specifies the procedure to be called.
  client_data Specifies additional data to be passed to the client’s event handler.
  position Specifies when the event handler is to be called relative to other previously registered handlers.

XtInsertEventHandler is identical to XtAddEventHandler with the additional position argument. If position is XtListHead, the event handler is registered so that it will be called before any event handlers that were previously registered for the same widget. If position is
XtListTail, the event handler is registered to be called after any previously registered event handlers. If the procedure is already registered with the same client_data value, the specified mask augments the existing mask and the procedure is repositioned in the list.

### 7.11.2. Event Handlers that Do Not Select Events

On occasion, clients need to register an event handler procedure with the dispatch mechanism without explicitly causing the X server to select for that event. To do this, use XtAddRawEventHandler.

```c
void XtAddRawEventHandler(w, event_mask, nonmaskable, proc, client_data)
    Widget w;
    EventMask event_mask;
    Boolean nonmaskable;
    XtEventHandler proc;
    XtPointer client_data;
```

- `w`: Specifies the widget for which this event handler is being registered. Must be of class Core or any subclass thereof.
- `event_mask`: Specifies the event mask for which to call this procedure.
- `nonmaskable`: Specifies whether this procedure should be called on the nonmaskable events (GraphicsExposure, NoExposure, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage, and MappingNotify).
- `proc`: Specifies the procedure to be called.
- `client_data`: Specifies additional data to be passed to the client’s event handler.

The XtAddRawEventHandler function is similar to XtAddEventHandler except that it does not affect the widget’s event mask and never causes an XSelectInput for its events. Note that the widget might already have those mask bits set because of other nonraw event handlers registered on it. If the procedure is already registered with the same client_data, the specified mask augments the existing mask. The order in which this procedure is called relative to other handlers registered for the same event is not defined.

To remove a previously registered raw event handler, use XtRemoveRawEventHandler.

```c
void XtRemoveRawEventHandler(w, event_mask, nonmaskable, proc, client_data)
    Widget w;
    EventMask event_mask;
    Boolean nonmaskable;
    XtEventHandler proc;
    XtPointer client_data;
```

- `w`: Specifies the widget for which this procedure is registered. Must be of class Core or any subclass thereof.
- `event_mask`: Specifies the event mask for which to unregister this procedure.
- `nonmaskable`: Specifies whether this procedure should be removed on the nonmaskable events (GraphicsExposure, NoExposure, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage, and MappingNotify).
- `proc`: Specifies the procedure to be registered.
- `client_data`: Specifies the registered client data.

The XtRemoveRawEventHandler function unregisters an event handler registered with XtAddRawEventHandler or XtInsertRawEventHandler for the specified events without changing the window event mask. The request is ignored if client_data does not match the value given when the handler was registered. If the specified procedure has not been
registered or if it has been registered with a different value of \textit{client\_data}, \texttt{XtRemoveRawEventHandler} returns without reporting an error.

To stop a procedure registered with \texttt{XtAddRawEventHandler} or \texttt{XtInsertRawEventHandler} from receiving all nonselected events, call \texttt{XtRemoveRawEventHandler} with an \textit{event\_mask} of \texttt{XtAllEvents} and \texttt{nonmaskable} \texttt{True}. The procedure will continue to receive any events that have been specified in calls to \texttt{XtAddEventHandler} or \texttt{XtInsertEventHandler}.

To register an event handler procedure that receives events before or after all previously registered event handlers without selecting for the events, use \texttt{XtInsertRawEventHandler}.

\begin{verbatim}
void XtInsertRawEventHandler(w, event_mask, nonmaskable, proc, client_data, position)

\end{verbatim}

\begin{itemize}
\item \texttt{w} Specifies the widget for which this event handler is being registered. Must be of class Core or any subclass thereof.
\item \texttt{event\_mask} Specifies the event mask for which to call this procedure.
\item \texttt{nonmaskable} Specifies whether this procedure should be called on the nonmaskable events (GraphicsExpose, NoExpose, SelectionClear, SelectionRequest, SelectionNotify, ClientMessage, and MappingNotify).
\item \texttt{proc} Specifies the procedure to be registered.
\item \texttt{client\_data} Specifies additional data to be passed to the client’s event handler.
\item \texttt{position} Specifies when the event handler is to be called relative to other previously registered handlers.
\end{itemize}

The \texttt{XtInsertRawEventHandler} function is similar to \texttt{XtInsertEventHandler} except that it does not modify the widget’s event mask and never causes an \texttt{XSelectInput} for the specified events. If the procedure is already registered with the same \texttt{client\_data} value, the specified mask augments the existing mask and the procedure is repositioned in the list.

\subsection{Current Event Mask}

To retrieve the event mask for a given widget, use \texttt{XtBuildEventMask}.

\begin{verbatim}
EventMask XtBuildEventMask(w)

\end{verbatim}

\begin{itemize}
\item \texttt{w} Specifies the widget. Must be of class Core or any subclass thereof.
\end{itemize}

The \texttt{XtBuildEventMask} function returns the event mask representing the logical OR of all event masks for event handlers registered on the widget with \texttt{XtAddEventHandler} and \texttt{XtInsertEventHandler} and all event translations, including accelerators, installed on the widget. This is the same event mask stored into the \texttt{XSetWindowAttributes} structure by \texttt{XtRealizeWidget} and sent to the server when event handlers and translations are installed or removed on the realized widget.
Applications and other widgets often need to register a procedure with a widget that gets called under certain prespecified conditions. For example, when a widget is destroyed, every procedure on the widget's `destroy_callbacks` list is called to notify clients of the widget's impending doom.

Every widget has an XtNdestroyCallbacks callback list resource. Widgets can define additional callback lists as they see fit. For example, the Pushbutton widget has a callback list to notify clients when the button has been activated.

Except where otherwise noted, it is the intent that all Intrinsics functions may be called at any time, including from within callback procedures, action routines, and event handlers.

### 8.1. Using Callback Procedure and Callback List Definitions

Callback procedure pointers for use in callback lists are of type `XtCallbackProc`.

```c
typedef void (*XtCallbackProc)(Widget, XtPointer, XtPointer);
```

* `w` Specifies the widget owning the list in which the callback is registered.
* `client_data` Specifies additional data supplied by the client when the procedure was registered.
* `call_data` Specifies any callback-specific data the widget wants to pass to the client. For example, when Scrollbar executes its XtNhumbChanged callback list, it passes the new position of the thumb.

The `client_data` argument provides a way for the client registering the callback procedure also to register client-specific data, for example, a pointer to additional information about the widget, a reason for invoking the callback, and so on. The `client_data` value may be NULL if all necessary information is in the widget. The `call_data` argument is a convenience to avoid having simple cases where the client could otherwise always call `XtGetValues` or a widget-specific function to retrieve data from the widget. Widgets should generally avoid putting complex state information in `call_data`. The client can use the more general data retrieval methods, if necessary.

Whenever a client wants to pass a callback list as an argument in an `XtCreateWidget`, `XtSetValues`, or `XtGetValues` call, it should specify the address of a NULL-terminated array of type `XtCallbackList`.

```c
typedef struct {
    XtCallbackProc callback;
    XtPointer closure;
} XtCallbackRec, *XtCallbackList;
```

For example, the callback list for procedures A and B with client data `clientDataA` and `clientDataB`, respectively, is
static XtCallbackRec callbacks[] = {
    {A, (XtPointer) clientDataA},
    {B, (XtPointer) clientDataB},
    {(XtCallbackProc) NULL, (XtPointer) NULL}
};

Although callback lists are passed by address in arglists and varargs lists, the Intrinsics recognize callback lists through the widget resource list and will copy the contents when necessary. Widget initialize and set_values procedures should not allocate memory for the callback list contents. The Intrinsics automatically do this, potentially using a different structure for their internal representation.

8.2. Identifying Callback Lists
Whenever a widget contains a callback list for use by clients, it also exports in its public .h file the resource name of the callback list. Applications and client widgets never access callback list fields directly. Instead, they always identify the desired callback list by using the exported resource name. All the callback manipulation functions described in this chapter except XtCallCallbackList check to see that the requested callback list is indeed implemented by the widget.

For the Intrinsics to find and correctly handle callback lists, they must be declared with a resource type of XtRCallback. The internal representation of a callback list is implementation-dependent; widgets may make no assumptions about the value stored in this resource if it is non-NULL. Except to compare the value to NULL (which is equivalent to XtCallbackStatus XtCallbackHasNone), access to callback list resources must be made through other Intrinsics procedures.

8.3. Adding Callback Procedures
To add a callback procedure to a widget's callback list, use XtAddCallback.

```c
void XtAddCallback(w, callback_name, callback, client_data)
    Widget w;
    String callback_name;
    XtCallbackProc callback;
    XtPointer client_data;
```

- `w` Specifies the widget. Must be of class Object or any subclass thereof.
- `callback_name` Specifies the callback list to which the procedure is to be appended.
- `callback` Specifies the callback procedure.
- `client_data` Specifies additional data to be passed to the specified procedure when it is invoked, or NULL.

A callback will be invoked as many times as it occurs in the callback list.

To add a list of callback procedures to a given widget's callback list, use XtAddCallbacks.

```c
void XtAddCallbacks(w, callback_name, callbacks)
    Widget w;
    String callback_name;
    XtCallbackList callbacks;
```

- `w` Specifies the widget. Must be of class Object or any subclass thereof.
- `callback_name` Specifies the callback list to which the procedures are to be appended.
callbacks Specifies the null-terminated list of callback procedures and corresponding client data.

8.4. Removing Callback Procedures
To delete a callback procedure from a widget's callback list, use XtRemoveCallback.

```c
void XtRemoveCallback(w, callback_name, callback, client_data)
  Widget w;
  String callback_name;
  XtCallbackProc callback;
  XtPointer client_data;
```

- `w` Specifies the widget. Must be of class Object or any subclass thereof.
- `callback_name` Specifies the callback list from which the procedure is to be deleted.
- `callback` Specifies the callback procedure.
- `client_data` Specifies the client data to match with the registered callback entry.

The XtRemoveCallback function removes a callback only if both the procedure and the client data match.

To delete a list of callback procedures from a given widget's callback list, use XtRemoveCallbacks.

```c
void XtRemoveCallbacks(w, callback_name, callbacks)
  Widget w;
  String callback_name;
  XtCallbackList callbacks;
```

- `w` Specifies the widget. Must be of class Object or any subclass thereof.
- `callback_name` Specifies the callback list from which the procedures are to be deleted.
- `callbacks` Specifies the null-terminated list of callback procedures and corresponding client data.

To delete all callback procedures from a given widget's callback list and free all storage associated with the callback list, use XtRemoveAllCallbacks.

```c
void XtRemoveAllCallbacks(w, callback_name)
  Widget w;
  String callback_name;
```

- `w` Specifies the widget. Must be of class Object or any subclass thereof.
- `callback_name` Specifies the callback list to be cleared.

8.5. Executing Callback Procedures
To execute the procedures in a given widget's callback list, specifying the callback list by resource name, use XtCallCallbacks.

```c
void XtCallCallbacks(w, callback_name, call_data)
  Widget w;
  String callback_name;
  XtPointer call_data;
```

- `w` Specifies the widget. Must be of class Object or any subclass thereof.
callback_name Specifies the callback list to be executed.
call_data Specifies a callback-list-specific data value to pass to each of the callback procedure in the list, or NULL.

XtCallCallbacks calls each of the callback procedures in the list named by callback_name in the specified widget, passing the client data registered with the procedure and call_data.

To execute the procedures in a callback list, specifying the callback list by address, use XtCallCallbackList.

void XtCallCallbackList(widget, callbacks, call_data)
   Widget widget;
   XtCallbackList callbacks;
   XtPointer call_data;

   widget Specifies the widget instance that contains the callback list. Must be of class Object or any subclass thereof.
callbacks Specifies the callback list to be executed.
call_data Specifies a callback-list-specific data value to pass to each of the callback procedures in the list, or NULL.

The callbacks parameter must specify the contents of a widget or object resource declared with representation type XtRCallback. If callbacks is NULL, XtCallCallbackList returns immediately; otherwise it calls each of the callback procedures in the list, passing the client data and call_data.

8.6. Checking the Status of a Callback List

To find out the status of a given widget’s callback list, use XtHasCallbacks.

typedef enum {XtCallbackNoList, XtCallbackHasNone, XtCallbackHasSome} XtCallbackStatus;

XtCallbackStatus XtHasCallbacks(w, callback_name)
   
   Widget w;
   String callback_name;

   w Specifies the widget. Must be of class Object or any subclass thereof.
callback_name Specifies the callback list to be checked.

The XtHasCallbacks function first checks to see if the widget has a callback list identified by callback_name. If the callback list does not exist, XtHasCallbacks returns XtCallbackNoList. If the callback list exists but is empty, it returns XtCallbackHasNone. If the callback list exists and has at least one callback registered, it returns XtCallbackHasSome.
Chapter 9
Resource Management

A resource is a field in the widget record with a corresponding resource entry in the resources list of the widget or any of its superclasses. This means that the field is settable by XtCreateWidget (by naming the field in the argument list), by an entry in a resource file (by using either the name or class), and by XtSetValues. In addition, it is readable by XtGetValues. Not all fields in a widget record are resources. Some are for bookkeeping use by the generic routines (like managed and being_destroyed). Others can be for local bookkeeping, and still others are derived from resources (many graphics contexts and pixmaps).

Widgets typically need to obtain a large set of resources at widget creation time. Some of the resources come from the argument list supplied in the call to XtCreateWidget, some from the resource database, and some from the internal defaults specified by the widget. Resources are obtained first from the argument list, then from the resource database for all resources not specified in the argument list, and last, from the internal default, if needed.

9.1. Resource Lists

A resource entry specifies a field in the widget, the textual name and class of the field that argument lists and external resource files use to refer to the field, and a default value that the field should get if no value is specified. The declaration for the XtResource structure is:

```c
typedef struct {
    String resource_name;
    String resource_class;
    String resource_type;
    Cardinal resource_size;
    Cardinal resource_offset;
    String default_type;
    XtPointer default_addr;
} XtResource, *XtResourceList;
```

When the resource list is specified as the CoreClassPart, ObjectClassPart, RectObjClassPart, or ConstraintClassPart resources field the strings pointed to by resource_name, resource_class, resource_type, and default_type must be permanently allocated prior to or during the execution of the class initialization procedure and must not be subsequently deallocated.

The resource_name field contains the name used by clients to access the field in the widget. By convention, it starts with a lower-case letter and is spelled exactly like the field name, except all underscores (_) are deleted and the next letter is replaced by its upper-case counterpart. For example, the resource name for background_pixel becomes backgroundPixel.

Resource names beginning with the two-character sequence “xt” and resource classes beginning with the two-character sequence “Xt” are reserved to the Intrinsics for future standard and implementation-dependent uses. Widget header files typically contain a symbolic name for each resource name. All resource names, classes, and types used by the Intrinsics are named in <X11/StringDefs.h>. The Intrinsics’s symbolic resource names begin with “XtN” and are followed by the string name (for example, XtNbackgroundPixel for backgroundPixel).
The `resource_class` field contains the class string used in resource specification files to identify the field. A resource class provides two functions:

- It isolates an application from different representations that widgets can use for a similar resource.
- It lets you specify values for several actual resources with a single name. A resource class should be chosen to span a group of closely related fields.

For example, a widget can have several pixel resources: background, foreground, border, block cursor, pointer cursor, and so on. Typically, the background defaults to white and everything else to black. The resource class for each of these resources in the resource list should be chosen so that it takes the minimal number of entries in the resource database to make the background offwhite and everything else darkblue.

In this case, the background pixel should have a resource class of "Background" and all the other pixel entries a resource class of "Foreground". Then, the resource file needs only two lines to change all pixels to offwhite or darkblue:

*Background: offwhite
*Foreground: darkblue

Similarly, a widget may have several font resources (such as normal and bold), but all fonts should have the class Font. Thus, changing all fonts simply requires only a single line in the default resource file:

*Font: 6x13

By convention, resource classes are always spelled starting with a capital letter to distinguish them from resource names. Their symbolic names are preceded with "XtC" (for example, XtCBackground).

The `resource_type` field gives the physical representation type of the resource and also encodes information about the specific usage of the field. By convention, it starts with an upper-case letter and is spelled identically to the type name of the field. The resource type is used when resources are fetched to convert from the resource database format (usually String) or the format of the resource default value (almost anything, but often String) to the desired physical representation (see Section 9.6). The Intrinsics define the following resource types:

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Structure or Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtRAcceleratorTable</td>
<td>XtAccelerators</td>
</tr>
<tr>
<td>XtRAtom</td>
<td>Atom</td>
</tr>
<tr>
<td>XtRBitmap</td>
<td>Xtpixmap, depth=1</td>
</tr>
<tr>
<td>XtRBoolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>XtRBool</td>
<td>Bool</td>
</tr>
<tr>
<td>XtRCallback</td>
<td>XtCallbackList</td>
</tr>
<tr>
<td>XtRCardinal</td>
<td>Cardinal</td>
</tr>
<tr>
<td>XtRColor</td>
<td>XColor</td>
</tr>
<tr>
<td>XtRColormap</td>
<td>XColor</td>
</tr>
<tr>
<td>XtRCursor</td>
<td>Cursor</td>
</tr>
<tr>
<td>XtRDimension</td>
<td>Dimension</td>
</tr>
<tr>
<td>XtRDisplay</td>
<td>Display*</td>
</tr>
<tr>
<td>XtREnum</td>
<td>XtEnum</td>
</tr>
<tr>
<td>XtRFile</td>
<td>FILE*</td>
</tr>
<tr>
<td>XtRFloat</td>
<td>float</td>
</tr>
<tr>
<td>XtRFont</td>
<td>Font</td>
</tr>
<tr>
<td>XtRFontSet</td>
<td>XFontSet</td>
</tr>
<tr>
<td>XtRFontStruct</td>
<td>XFontStruct*</td>
</tr>
</tbody>
</table>
### Resource Type vs. Structure or Field Type

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Structure or Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtRFunction</td>
<td>(*)()</td>
</tr>
<tr>
<td>XtRGeometry</td>
<td>char*, format as defined by XPar-</td>
</tr>
<tr>
<td>XtrInitialState</td>
<td>int</td>
</tr>
<tr>
<td>XtrInt</td>
<td>int</td>
</tr>
<tr>
<td>XtrLongBoolean</td>
<td>long</td>
</tr>
<tr>
<td>XtrObject</td>
<td>Object</td>
</tr>
<tr>
<td>XtrPixel</td>
<td>Pixel</td>
</tr>
<tr>
<td>XtrPixmap</td>
<td>Pixmap</td>
</tr>
<tr>
<td>XtrPointer</td>
<td>XtPointer</td>
</tr>
<tr>
<td>XtrPosition</td>
<td>Position</td>
</tr>
<tr>
<td>XtrScreen</td>
<td>Screen*</td>
</tr>
<tr>
<td>XtrShort</td>
<td>short</td>
</tr>
<tr>
<td>XtrString</td>
<td>String</td>
</tr>
<tr>
<td>XtrStringArray</td>
<td>String*</td>
</tr>
<tr>
<td>XtrStringTable</td>
<td>String*</td>
</tr>
<tr>
<td>XtrTranslationTable</td>
<td>XtTranslations</td>
</tr>
<tr>
<td>XtrUnsignedChar</td>
<td>unsigned char</td>
</tr>
<tr>
<td>XtrVisual</td>
<td>Visual*</td>
</tr>
<tr>
<td>XtrWidget</td>
<td>Widget</td>
</tr>
<tr>
<td>XtrWidgetClass</td>
<td>WidgetClass</td>
</tr>
<tr>
<td>XtrWidgetList</td>
<td>WidgetList</td>
</tr>
<tr>
<td>XtrWindow</td>
<td>Window</td>
</tr>
</tbody>
</table>

<X11/StringDefs.h> also defines the following resource types as a convenience for widgets, although they do not have any corresponding data type assigned: XtREditMode, XtRJustify, and XtROrientation.

The `resource_size` field is the size of the physical representation in bytes; you should specify it as `sizeof(type)` so that the compiler fills in the value. The `resource_offset` field is the offset in bytes of the field within the widget. You should use the `XtOffsetOf` macro to retrieve this value. The `default_type` field is the representation type of the default resource value. If `default_type` is different from `resource_type` and the default value is needed, the resource manager invokes a conversion procedure from `default_type` to `resource_type`. Whenever possible, the default type should be identical to the resource type in order to minimize widget creation time. However, there are sometimes no values of the type that the program can easily specify. In this case, it should be a value for which the converter is guaranteed to work (for example, `XtDefaultForeground` for a pixel resource). The `default_addr` field specifies the address of the default resource value. As a special case, if `default_type` is `XtrString`, then the value in the `default_addr` field is the pointer to the string rather than a pointer to the pointer. The default is used if a resource is not specified in the argument list or in the resource database, or if the conversion from the representation type stored in the resource database fails, which can happen for various reasons (for example, a misspelled entry in a resource file).

Two special representation types (XtRImmediate and XtRCallProc) are usable only as default resource types. XtRImmediate indicates that the value in the `default_addr` field is the actual value of the resource rather than the address of the value. The value must be in the correct representation type for the resource, coerced to an `XtPointer`. No conversion is possible, since there is no source representation type. XtRCallProc indicates that the value in the `default_addr` field is a procedure pointer. This procedure is automatically invoked with the widget, `resource_offset`, and a pointer to an `XrmValue` in which to store the result.

XtRCallProc procedure pointers are of type `XtResourceDefaultProc`.

110
typedef void (*XtResourceDefaultProc)(Widget, int, XrmValue*);

    Widget w;
    int offset;
    XrmValue *value;

w Specifies the widget whose resource value is to be obtained.

offset Specifies the offset of the field in the widget record.

value Specifies the resource value descriptor to return.

The XtResourceDefaultProc procedure should fill in the value->addr field with a pointer to the resource value in its correct representation type.

To get the resource list structure for a particular class, use XtGetResourceList.

void XtGetResourceList(char?, class, resources_return, num_resources_return);

    WidgetClass class;
    XtResourceList *resources_return;
    Cardinal *num_resources_return;

class Specifies the object class to be queried. It must be objectClass or any subclass thereof.

resources_return Returns the resource list.

num_resources_return Returns the number of entries in the resource list.

If XtGetResourceList is called before the class is initialized, it returns the resource list as specified in the class record. If it is called after the class has been initialized, XtGetResourceList returns a merged resource list that includes the resources for all superclasses. The list returned by XtGetResourceList should be freed using XtFree when it is no longer needed.

To get the constraint resource list structure for a particular widget class, use XtGetConstraintResourceList.

void XtGetConstraintResourceList(class, resources_return, num_resources_return);

    WidgetClass class;
    XtResourceList *resources_return;
    Cardinal *num_resources_return;

class Specifies the object class to be queried. It must be objectClass or any subclass thereof.

resources_return Returns the constraint resource list.

num_resources_return Returns the number of entries in the constraint resource list.

If XtGetConstraintResourceList is called before the widget class is initialized, the resource list as specified in the widget class Constraint part is returned. If XtGetConstraintResourceList is called after the widget class has been initialized, the merged resource list for the class and all Constraint superclasses is returned. If the specified class is not a subclass of constraintWidgetClass, *resources_return is set to NULL and *num_resources_return is set to zero. The list returned by XtGetConstraintResourceList should be freed using XtFree when it is no longer needed.

The routines XtSetValues and XtGetValues also use the resource list to set and get widget state; see Sections 9.7.1 and 9.7.2.

Here is an abbreviated version of a possible resource list for a Label widget:

    /* Resources specific to Label */
    static XtResource resources[] = {
        [XtNforeground, XtCForeground, XtRPixel, sizeof(Pixel),

111
The complete resource name for a field of a widget instance is the concatenation of the application shell name (from XtAppCreateShell), the instance names of all the widget’s parents up to the top of the widget tree, the instance name of the widget itself, and the resource name of the specified field of the widget. Similarly, the full resource class of a field of a widget instance is the concatenation of the application class (from XtAppCreateShell), the widget class names of all the widget’s parents up to the top of the widget tree, the widget class name of the widget itself, and the resource class of the specified field of the widget.

9.2. Byte Offset Calculations
To determine the byte offset of a field within a structure type, use XtOffsetOf.

Cardinal XtOffsetOf(structure_type, field_name)

Type structure_type;
Field field_name;

structure_type Specifies a type that is declared as a structure.
field_name Specifies the name of a member within the structure.

The XtOffsetOf macro expands to a constant expression that gives the offset in bytes to the specified structure member from the beginning of the structure. It is normally used to statically initialize resource lists and is more portable than XtOffset, which serves the same function.

To determine the byte offset of a field within a structure pointer type, use XtOffset.

Cardinal XtOffset(pointer_type, field_name)

Type pointer_type;
Field field_name;

pointer_type Specifies a type that is declared as a pointer to a structure.
field_name Specifies the name of a member within the structure.

The XtOffset macro expands to a constant expression that gives the offset in bytes to the specified structure member from the beginning of the structure. It may be used to statically initialize resource lists. XtOffset is less portable than XtOffsetOf.

9.3. Superclass-to-Subclass Chaining of Resource Lists
The XtCreateWidget function gets resources as a superclass-to-subclass chained operation. That is, the resources specified in the objectClass resource list are fetched, then those in rectObjClass, and so on down to the resources specified for this widget’s class. Within a class, resources are fetched in the order they are declared.

In general, if a widget resource field is declared in a superclass, that field is included in the superclass’s resource list and need not be included in the subclass’s resource list. For example, the Core class contains a resource entry for background_pixel. Consequently, the implementation of Label need not also have a resource entry for background_pixel. However, a subclass, by specifying a resource entry for that field in its own resource list, can override the
resource entry for any field declared in a superclass. This is most often done to override the
defaults provided in the superclass with new ones. At class initialization time, resource lists
for that class are scanned from the superclass down to the class to look for resources with the
same offset. A matching resource in a subclass will be reordered to override the superclass
entry. If reordering is necessary, a copy of the superclass resource list is made to avoid
affecting other subclasses of the superclass.

Also at class initialization time, the Intrinsics produce an internal representation of the resource
list to optimize access time when creating widgets. In order to save memory, the Intrinsics
may overwrite the storage allocated for the resource list in the class record; therefore, widgets
must allocate resource lists in writable storage and must not access the list contents directly
after the class_initialize procedure has returned.

9.4. Subresources

A widget does not do anything to retrieve its own resources; instead, XtCreateWidget does
this automatically before calling the class initialize procedure.

Some widgets have subparts that are not widgets but for which the widget would like to fetch
resources. Such widgets call XtGetSubresources to accomplish this.

```c
void XtGetSubresources(w, base, name, class, resources, num_resources, args, num_args)
```

- `w` Specifies the object used to qualify the subpart resource name and class. Must
  be of class Object or any subclass thereof.
- `base` Specifies the base address of the subpart data structure into which the resources
  will be written.
- `name` Specifies the name of the subpart.
- `class` Specifies the class of the subpart.
- `resources` Specifies the resource list for the subpart.
- `num_resources` Specifies the number of entries in the resource list.
- `args` Specifies the argument list to override any other resource specifications.
- `num_args` Specifies the number of entries in the argument list.

The XtGetSubresources function constructs a name and class list from the application name
and class, the names and classes of all the object’s ancestors, and the object itself. Then it
appends to this list the name and class pair passed in. The resources are fetched from the
argument list, the resource database, or the default values in the resource list. Then they are
copied into the subpart record. If args is NULL, num_args must be zero. However, if
num_args is zero, the argument list is not referenced.

XtGetSubresources may overwrite the specified resource list with an equivalent representation
in an internal format, which optimizes access time if the list is used repeatedly. The resource
list must be allocated in writable storage, and the caller must not modify the list contents after
the call if the same list is to be used again. Resources fetched by XtGetSubresources are
reference-counted as if they were referenced by the specified object. Subresources might there¬
fore be freed from the conversion cache and destroyed when the object is destroyed, but not
before then.
To fetch resources for widget subparts using varargs lists, use `XtVaGetSubresources`.

```c
void XtVaGetSubresources(w, base, name, class, resources, num_resources, ...)

    Widget w;
    XtPointer base;
    String name;
    String class;
    XtResourceList resources;
    Cardinal num_resources;
```

- **w** Specifies the object used to qualify the subpart resource name and class. Must be of class Object or any subclass thereof.
- **base** Specifies the base address of the subpart data structure into which the resources will be written.
- **name** Specifies the name of the subpart.
- **class** Specifies the class of the subpart.
- **resources** Specifies the resource list for the subpart.
- **num_resources** Specifies the number of entries in the resource list.
- **...** Specifies the variable argument list to override any other resource specifications.

`XtVaGetSubresources` is identical in function to `XtGetSubresources` with the `args` and `num_args` parameters replaced by a varargs list, as described in Section 2.5.1.

### 9.5. Obtaining Application Resources

To retrieve resources that are not specific to a widget but apply to the overall application, use `XtGetApplicationResources`.

```c
void XtGetApplicationResources(w, base, resources, num_resources, args, num_args)

    Widget w;
    XtPointer base;
    XtResourceList resources;
    Cardinal num_resources;
    ArgList args;
    Cardinal num_args;
```

- **w** Specifies the object that identifies the resource database to search (the database is that associated with the display for this object). Must be of class Object or any subclass thereof.
- **base** Specifies the base address into which the resource values will be written.
- **resources** Specifies the resource list.
- **num_resources** Specifies the number of entries in the resource list.
- **args** Specifies the argument list to override any other resource specifications.
- **num_args** Specifies the number of entries in the argument list.

The `XtGetApplicationResources` function first uses the passed object, which is usually an application shell widget, to construct a resource name and class list. The full name and class of the specified object (that is, including its ancestors, if any) is logically added to the front of each resource name and class. Then it retrieves the resources from the argument list, the resource database, or the resource list default values. After adding base to each address, `XtGetApplicationResources` copies the resources into the addresses obtained by adding `base` to each offset in the resource list. If `args` is NULL, `num_args` must be zero. However, if `num_args` is zero, the argument list is not referenced. The portable way to specify application resources is to declare them as members of a structure and pass the address of the structure as
the base argument. 'Xi' Toolkit Intrinsics

The XtGetApplicationResources may overwrite the specified resource list with an equivalent representation in an internal format, which optimizes access time if the list is used repeatedly. The resource list must be allocated in writable storage, and the caller must not modify the list contents after the call if the same list is to be used again. Any per-display resources fetched by XtGetApplicationResources will not be freed from the resource cache until the display is closed.

To retrieve resources for the overall application using varargs lists, use XtVaGetApplicationResources.

```c
void XtVaGetApplicationResourcesOv,
    w, base, resources, num_resources, ...
    ...
```

- `w` Specifies the object that identifies the resource database to search (the database is that associated with the display for this object). Must be of class Object or any subclass thereof.
- `base` Specifies the base address into which the resource values will be written.
- `resources` Specifies the resource list for the subpart.
- `num_resources` Specifies the number of entries in the resource list.
- `...` Specifies the variable argument list to override any other resource specifications.

XtVaGetApplicationResources is identical in function to XtGetApplicationResources with the args and num_args parameters replaced by a varargs list, as described in Section 2.5.1.

### 9.6. Resource Conversions

The Intrinsics provide a mechanism for registering representation converters that are automatically invoked by the resource-fetching routines. The Intrinsics additionally provide and register several commonly used converters. This resource conversion mechanism serves several purposes:

- It permits user and application resource files to contain textual representations of nontextual values.
- It allows textual or other representations of default resource values that are dependent on the display, screen, or colormap, and thus must be computed at runtime.
- It caches conversion source and result data. Conversions that require much computation or space (for example, string-to-translation-table) or that require round-trips to the server (for example, string-to-font or string-to-color) are performed only once.

#### 9.6.1. Predefined Resource Converters

The Intrinsics define all the representations used in the Object, RectObj, Core, Composite, Constraint, and Shell widget classes. The Intrinsics register the following resource converters that accept input values of representation type XtRString.

<table>
<thead>
<tr>
<th>Target Representation</th>
<th>Converter Name</th>
<th>Additional Args</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtRAcceleratorTable</td>
<td>XtCvtStringToAcceleratorTable</td>
<td></td>
</tr>
</tbody>
</table>

115
The String-to-Pixel conversion has two predefined constants that are guaranteed to work and contrast with each other: XtDefaultForeground and XtDefaultBackground. They evaluate to the black and white pixel values of the widget's screen, respectively. If the application resource reverseVideo is True, they evaluate to the white and black pixel values of the widget's screen, respectively. Similarly, the String-to-Font and String-to-FontStruct converters recognize the constant XtDefaultFont and evaluate this in the following manner:

- Query the resource database for the resource whose full name is "XtDefaultFont", class "XtDefaultFont" (that is, no widget name/class prefixes) and use a type XtRString value returned as the font name, or a type XtRFont or XtRFontStruct value directly as the resource value.
- If the resource database does not contain a value for XtDefaultFont, class XtDefaultFont, or if the returned font name cannot be successfully opened, an implementation-defined font in ISO8859-1 character set encoding is opened. (One possible algorithm is to perform an XListFonts using a wildcard font name and use the first font in the list. This wildcard font name should be as broad as possible to maximize the probability of locating a useable font; for example, "*.**-R-**-**-**-120-**-**-**-ISO8859-1").
- If no suitable ISO8859-1 font can be found, issue a warning message and return False.

The String-to-FontSet converter recognizes the constant XtDefaultFontSet and evaluate this in the following manner:

- Query the resource database for the resource whose full name is "XtDefaultFontSet", class "XtDefaultFontSet" (that is, no widget name/class prefixes) and use a type XtRString value returned as the base font name list, or a type XtRFontSet value directly as the resource value.
- If the resource database does not contain a value for XtDefaultFontSet, class XtDefaultFontSet, or if a font set cannot be successfully created from this resource, an implementation-defined font set is created. (One possible algorithm is to perform an XCreateFontSet using a wildcard base font name. This wildcard base font name should be as broad as possible to maximize the probability of locating a useable font; for example, "*.**-R-**-**-**-120-**-**-**-**-**-ISO8859-1").
- If no suitable font set can be created, issue a warning message and return False.
If a font set is created but *missing_charset_list* is not empty, a warning is issued and the partial font set is returned. The Intrinsics register the String-to-FontSet converter with a conversion argument list that extracts the current process locale at the time the converter is invoked. This ensures that the converter is invoked again if the same conversion is required in a different locale.

The String-to-InitialState conversion accepts the values *NormalState* or *IconicState* as defined by the *Inter-Client Communication Conventions Manual*.

The String-to-Visual conversion calls *XMatchVisualInfo* using the screen and depth fields from the core part and returns the first matching Visual on the list. The widget resource list must be certain to specify any resource of type *XtRVisual* after the depth resource. The allowed string values are the visual class names defined in *X Window System Protocol*, Section 8; *StaticGray*, *StaticColor*, *TrueColor*, *GrayScale*, *PseudoColor*, and *DirectColor*.

The Intrinsics register the following resource converter that accepts an input value of representation type *XtRColor*.

<table>
<thead>
<tr>
<th>Target Representation</th>
<th>Converter Name</th>
<th>Additional Args</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtRPixel</td>
<td>XtCvtColorToPixel</td>
<td></td>
</tr>
</tbody>
</table>

The Intrinsics register the following resource converters that accept input values of representation type *XtRInt*.

<table>
<thead>
<tr>
<th>Target Representation</th>
<th>Converter Name</th>
<th>Additional Args</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtRBoolean</td>
<td>XtCvtIntToBoolean</td>
<td></td>
</tr>
<tr>
<td>XtRBool</td>
<td>XtCvtIntToBool</td>
<td></td>
</tr>
<tr>
<td>XtRColor</td>
<td>XtCvtIntToColor</td>
<td>colorConvertArgs</td>
</tr>
<tr>
<td>XtRDimension</td>
<td>XtCvtIntToDimension</td>
<td></td>
</tr>
<tr>
<td>XtRFloat</td>
<td>XtCvtIntToFloat</td>
<td></td>
</tr>
<tr>
<td>XtRFont</td>
<td>XtCvtIntToFont</td>
<td></td>
</tr>
<tr>
<td>XtRPixel</td>
<td>XtCvtIntToPixel</td>
<td></td>
</tr>
<tr>
<td>XtRPixmap</td>
<td>XtCvtIntToPixmap</td>
<td></td>
</tr>
<tr>
<td>XtRPosition</td>
<td>XtCvtIntToPosition</td>
<td></td>
</tr>
<tr>
<td>XtRShort</td>
<td>XtCvtIntToShort</td>
<td></td>
</tr>
<tr>
<td>XtRUnsignedChar</td>
<td>XtCvtIntToUnsignedChar</td>
<td></td>
</tr>
</tbody>
</table>

The Intrinsics register the following resource converter that accepts an input value of representation type *XtRPixel*.

<table>
<thead>
<tr>
<th>Target Representation</th>
<th>Converter Name</th>
<th>Additional Args</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtRColor</td>
<td>XtCvtPixelToColor</td>
<td></td>
</tr>
</tbody>
</table>
9.6.2. New Resource Converters

Type converters use pointers to XrmValue structures (defined in <X11/Xresource.h>; see Section 15.4 in Xlib – C Language X Interface) for input and output values.

typedef struct {
    unsigned int size;
    XPointer addr;
} XrmValue, *XrmValuePtr;

The addr field specifies the address of the data and the size field gives the total number of significant bytes in the data. For values of type String, addr is the address of the first character and size includes the NUL terminating byte.

A resource converter procedure pointer is of type XtTypeConverter.

typedef Boolean (*XtTypeConverter)(Display*, XrmValue*, Cardinal*,
XrmValue*, XrmValue*, XtPointer*);

Display *display;
XrmValue *args;
Cardinal *num_args;
XrmValue *from;
XrmValue *to;
XtPointer *converter_data;

display Specifies the display connection with which this conversion is associated.
args Specifies a list of additional XrmValue arguments to the converter if additional context is needed to perform the conversion, or NULL. For example, the String-to-Font converter needs the widget’s screen, and the String-to-Pixel converter needs the widget’s screen and colormap.
num_args Specifies the number of entries in args.
from Specifies the value to convert.
to Specifies a descriptor for a location into which to store the converted value.
converter_data Specifies a location into which the converter may store converter-specific data associated with this conversion.

The display argument is normally used only when generating error messages, to identify the application context (with the function XtDisplayToApplicationContext).

The to argument specifies the size and location into which the converter should store the converted value. If the addr field is NULL, the converter should allocate appropriate storage and store the size and location into the to descriptor. If the type converter allocates the storage, it remains under the ownership of the converter and must not be modified by the caller. The type converter is permitted to use static storage for this purpose, and therefore the caller must immediately copy the data upon return from the converter. If the addr field is not NULL, the converter must check the size field to ensure that sufficient space has been allocated before storing the converted value. If insufficient space is specified, the converter should update the size field with the number of bytes required and return False without modifying the data at the specified location. If sufficient space was allocated by the caller, the converter should update the size field with the number of bytes actually occupied by the converted value. For converted values of type XtRString, the size should include the NULL terminating byte, if any. The converter may store any value in the location specified in converter_data; this data will be passed to the destructor, if any, when the resource is freed by the Intrinsics.

The converter must return True if the conversion was successful and False otherwise. If the conversion cannot be performed because of an improper source value, a warning message should also be issued with XtAppWarningMsg.
Most type converters just take the data described by the specified from argument and return data by writing into the location specified in the to argument. A few need other information, which is available in args. A type converter can invoke another type converter, which allows differing sources that may convert into a common intermediate result to make maximum use of the type converter cache.

Note that if an address is written into to->addr, it cannot be that of a local variable of the converter because the data will not be valid after the converter returns. Static variables may be used, as in the following example. If the converter modifies the resource database, the changes affect any in-progress widget creation, XtGetApplicationResources, or XtGetSubresources in an implementation-defined manner; however, insertion of new entries or changes to existing entries is allowed and will not directly cause an error.

The following is an example of a converter that takes a string and converts it to a Pixel.
Note that the display parameter is only used to generate error messages; the Screen conversion argument is still required to inform the Intrinsics that the converted value is a function of the particular display (and colormap).

```c
#define done(type, value) \
{ \
    if (toVal->addr != NULL) { \
        if (toVal->size < sizeof(type)) { \
            toVal->size = sizeof(type); \
            return False; \
        } \
        *(type*)(toVal->addr) = (value); \
    } \
    else { \
        static type static_val; \
        static_val = (value); \
        toVal->addr = (XPointer)&static_val; \
    } \
    toVal->size = sizeof(type); \
    return True; \
}
```

```c
static Boolean CvtStringToPixel(dpy, args, num_args, fromVal, toVal, converier_data) 
  Display *dpy; 
  XrmValue *args; 
  Cardinal *num_args; 
  XrmValue *fromVal; 
  XrmValue *toVal; 
  XtPointer *converter_data; 
{
  static XColor screenColor; 
  XColor exactColor; 
  Screen *screen; 
  Colormap colormap; 
  Status status; 
  char message[1000];

  if (*num_args != 2) 
    XtAppErrorMsg(XtDisplayToApplicationContcxt(dpy), 
      "cvtStringToPixel", "wrongParameters", "XiToolkitError", 
      "String to pixel conversion needs screen and colormap arguments", 
      119
```
(String *)NULL, (Cardinal *)NULL);

screen = *((Screen**) args[0].addr);
colormap = *((Colormap *) args[1].addr);

LowerCase((char *) fromVal->addr, message);

if (strcmp(message, "xtdefaultbackground") == 0) done(&WhitePixelOfScreen(screen), Pixel);
if (strcmp(message, "xtdefaultforeground") == 0) donc(&BlackPixelOfScreen(screen), Pixel);

status = XAllocNamedColor(DisplayOfScreen(screen), colormap, (char*)fromVal->addr,
&screenColor, &exactColor);

if (status == 0) {
    String params[1];
    Cardinal num_params = 1;
    params[0] = (String)fromVal->addr;
    XtAppWarningMsg(XtDisplayToApplicationContext(dpy),
    "cvtStringToPixel", "noColormap", "XtToolkitError",
    "Cannot allocate colormap entry for \"%s\"", params, &num_params);
} else {
    done( &screenColor.pixel, Pixel );
}

/* converter_data not used here */
};

All type converters should define some set of conversion values for which they are guaranteed
to succeed so these can be used in the resource defaults. This issue arises only with conver¬
sions, such as fonts and colors, where there is no string representation that all server imple¬
mentations will necessarily recognize. For resources like these, the converter should define a
symbolic constant in the same manner as XtDefaultForeground, XtDefaultBackground, and
XtDefaultFont.

To allow the Intrinsics to deallocate resources produced by type converters, a resource destruc¬
tor procedure may also be provided.

A resource destructor procedure pointer is of type XtDestructor.

typedef void (*XtDestructor) (XtAppContext, XrmValue*, XtPointer, XrmValue*, Cardinal*);

XtAppContext app;
XrmValue *to;
XtPointer converter_data;
XrmValue *args;
Cardinal *num_args;

app Specifies an application context in which the resource is being freed.
to Specifies a descriptor for the resource produced by the type converter.
converter_data Specifies the converter-specific data returned by the type converter.
args Specifies the additional converter arguments as passed to the type converter
    when the conversion was performed.
um_args Specifies the number of entries in args.

The destructor procedure is responsible for freeing the resource specified by the to argument,
including any auxiliary storage associated with that resource, but not the memory directly
addressed by the size and location in the to argument nor the memory specified by args.

120
9.6.3. Issuing Conversion Warnings

The XtDisplayStringConversionWarning procedure is a convenience routine for resource type converters that convert from string values.

```c
void XtDisplayStringConversionWarning(display, from_value, to_type);
```

- `display`: Specifies the display connection with which the conversion is associated.
- `from_value`: Specifies the string that could not be converted.
- `to_type`: Specifies the target representation type requested.

The XtDisplayStringConversionWarning procedure issues a warning message using XtAppWarningMsg with `name``conversionError```, `type``string```, `class``XtToolkitError```, and the default message ``Cannot convert ``from_value`` to type `to_type````.

To issue other types of warning or error messages, the type converter should use XtAppWarningMsg or XtAppErrorMsg.

To retrieve the application context associated with a given display connection, use XtDisplayToApplicationContext.

```c
XtAppContext XtDisplayToApplicationContext( display )
```

- `display`: Specifies an open and initialized display connection.

The XtDisplayToApplicationContext function returns the application context in which the specified `display` was initialized. If the display is not known to the Intrinsics, an error message is issued.

9.6.4. Registering a New Resource Converter

When registering a resource converter, the client must specify the manner in which the conversion cache is to be used when there are multiple calls to the converter. Conversion cache control is specified via an XtCacheType argument.

```c
typedef int XtCacheType;
```

An XtCacheType field may contain one of the following values:

- **XtCacheNone**
  - Specifies that the results of a previous conversion may not be reused to satisfy any other resource requests; the specified converter will be called each time the converted value is required.

- **XtCacheAll**
  - Specifies that the results of a previous conversion should be reused for any resource request that depends upon the same source value and conversion arguments.

- **XtCacheByDisplay**
  - Specifies that the results of a previous conversion should be used as for XtCacheAll but the destructor will be called, if specified, if XtCloseDisplay is called for the display connection associated with the converted value, and the value will be removed from the conversion cache.
The qualifier XtCacheRefCount may be ORed with any of the above values. If XtCacheRefCount is specified, calls to XtCreateWidget, XtCreateManagedWidget, XtGetApplicationResources and XtGetSubresources that use the converted value will be counted. When a widget using the converted value is destroyed, the count is decremented, and if the count reaches zero, the destructor procedure will be called and the converted value will be removed from the conversion cache.

To register a type converter for all application contexts in a process, use XtSetTypeConverter and to register a type converter in a single application context, use XtAppSetTypeConverter:

```c
void XtSetTypeConverter(String from_type, String to_type, XtTypeConverter converter, XtConvertArgList convert_args, Cardinal num_args, String cache_type, XtDestructor destructor);
```

- `from_type`: Specifies the source type.
- `to_type`: Specifies the destination type.
- `converter`: Specifies the resource type converter procedure.
- `convert_args`: Specifies additional conversion arguments, or NULL.
- `num_args`: Specifies the number of entries in `convert_args`.
- `cache_type`: Specifies whether or not resources produced by this converter are sharable or display-specific and when they should be freed.
- `destructor`: Specifies a destroy procedure for resources produced by this conversion, or NULL if no additional action is required to deallocate resources produced by the converter.

```c
void XtAppSetTypeConverter(XtAppContext app_context, String from_type, String to_type, XtTypeConverter converter, XtConvertArgList convert_args, Cardinal num_args, String cache_type, XtDestructor destructor);
```

- `app_context`: Specifies the application context.
- `from_type`: Specifies the source type.
- `to_type`: Specifies the destination type.
- `converter`: Specifies the resource type converter procedure.
- `convert_args`: Specifies additional conversion arguments, or NULL.
- `num_args`: Specifies the number of entries in `convert_args`.
- `cache_type`: Specifies whether or not resources produced by this converter are sharable or display-specific and when they should be freed.
Specifies a destroy procedure for resources produced by this conversion, or NULL if no additional action is required to deallocate resources produced by the converter.

XtSetTypeConverter registers the specified type converter and destructor in all application contexts created by the calling process, including any future application contexts that may be created. XtAppSetTypeConverter registers the specified type converter in the single application context specified. If the same from_type and to_type are specified in multiple calls to either function, the most recent overrides the previous ones.

For the few type converters that need additional arguments, the Intrinsics conversion mechanism provides a method of specifying how these arguments should be computed. The enumerated type XtAddressMode and the structure XtConvertArgRec specify how each argument is derived. These are defined in <X11/Intrinsic.h>.

typedef enum {
    /* address mode         parameter representation */
    XtAddress,    /* address */
    XtBaseOffset, /* offset */
    XtImmediate,  /* constant */
    XtResourceString, /* resource name string */
    XtResourceQuark,           /* resource name quark */
    XtWidgetBaseOffset, /* offset */
    XtProcedureArg            /* procedure to call */
} XtAddressMode;

typedef struct {
    XtAddressMode address_mode;
    XtPointer address_id;
    Cardinal size;
} XtConvertArgRec, *XtConvertArgList;

The size field specifies the length of the data in bytes. The address_mode field specifies how the address_id field should be interpreted. XtAddress causes address_id to be interpreted as the address of the data. XtBaseOffset causes address_id to be interpreted as the offset from the widget base. XtImmediate causes address_id to be interpreted as a constant. XtResourceString causes address_id to be interpreted as the name of a resource that is to be converted into an offset from the widget base. XtResourceQuark causes address_id to be interpreted as the result of an XrmStringToQuark conversion on the name of a resource, which is to be converted into an offset from the widget base. XtWidgetBaseOffset is similar to XtBaseOffset except that it searches for the closest windowed ancestor if the object is not of a subclass of Core (See Chapter 12). XtProcedureArg specifies that address_id is a pointer to a procedure to be invoked to return the conversion argument. If XtProcedureArg is specified, address_id must contain the address of a function of type XtConvertArgProc.

typedef void (*XtConvertArgProc)(Widget, Cardinal*, XrmValue*);

void (*XtConvertArgProc)(Widget, Cardinal*, XrmValue*);

object Passes the object for which the resource is being converted, or NULL if the converter was invoked by XtCallConverter or XtDirectConvert.

size Passes a pointer to the size field from the XtConvertArgRec.

value Passes a pointer to a descriptor into which the procedure must store the conversion argument.
When invoked, the \texttt{XtConvertArgProc} procedure must derive a conversion argument and store the address and size of the argument in the location pointed to by \texttt{value}.

In order to permit reentrancy, the \texttt{XtConvertArgProc} should return the address of storage whose lifetime is no shorter than the lifetime of \texttt{object}. If \texttt{object} is \texttt{NULL}, the lifetime of the conversion argument must be no shorter than the lifetime of the resource with which the conversion argument is associated. The Intrinsics do not guarantee to copy this storage but do guarantee not to reference it if the resource is removed from the conversion cache.

The following example illustrates how to register the \texttt{CvtStringToPixel} routine given earlier:

```c
static XtConvertArgRec colorConvertArgs[] = {
    {XtWidgetBaseOffset, (XtPointer) XtOffset(Widget, core.screen), sizeof(Screen*)},
    {XtWidgetBaseOffset, (XtPointer) XtOffset(Widget, core.colormap), sizeof(Colormap)}
};

XtSetTypeConverter(XtRString, XtRPixel, CvtStringToPixel, colorConvertArgs, XtNumber(colorConvertArgs), XtCacheByDisplay, NULL);
```

The conversion argument descriptors \texttt{colorConvertArgs} and \texttt{screenConvertArg} are predefined by the Intrinsics. Both take the values from the closest windowed ancestor if the object is not of a subclass of Core. The \texttt{screenConvertArg} descriptor puts the widget's \texttt{screen} field into \texttt{args[0]}. The \texttt{colorConvertArgs} descriptor puts the widget's \texttt{screen} field into \texttt{args[0]}, and the widget's \texttt{colormap} field into \texttt{args[1]}.

Conversion routines should not just put a descriptor for the address of the base of the widget into \texttt{args[0]} and use that in the routine. They should pass in the actual values on which the conversion depends on. By keeping the dependencies of the conversion procedure specific, it is more likely that subsequent conversions will find what they need in the conversion cache. This way the cache is smaller and has fewer and more widely applicable entries.

If any conversion arguments of type \texttt{XtBaseOffset}, \texttt{XtResourceString}, \texttt{XtResourceQuark}, and \texttt{XtWidgetBaseOffset} are specified for conversions performed by \texttt{XtGetApplicationResources}, \texttt{XtGetSubresources}, \texttt{XtVaGetApplicationResources} or \texttt{XtVaGetSubresources}, the arguments are computed with respect to the specified widget, not the base address or resource list specified in the call.

If the \texttt{XtConvertArgProc} modifies the resource database, the changes affect any in-progress widget creation, \texttt{XtGetApplicationResources}, or \texttt{XtGetSubresources} in an implementation-defined manner; however, insertion of new entries or changes to existing entries is allowed and will not directly cause an error.

9.6.5. Resource Converter Invocation

All resource-fetching routines (for example, \texttt{XtGetSubresources}, \texttt{XtGetApplicationResources}, and so on) call resource converters if the resource database or varargs list specifies a value that has a different representation from the desired representation or if the widget's default resource value representation is different from the desired representation.

To invoke explicit resource conversions, use \texttt{XtConvertAndStore} or \texttt{XtCallConverter}.

typedef XtPointer XtCacheRef;
Boolean XtCallConverter(display, converter, conversion_args, num_args, from, to_in_out, cache_ref_return);

- **display** specifies the display with which the conversion is to be associated.
- **converter** specifies the conversion procedure to be called.
- **conversion_args** specifies the additional conversion arguments needed to perform the conversion, or NULL.
- **num_args** specifies the number of entries in **conversion_args**.
- **from** specifies a descriptor for the source value.
- **to_in_out** returns the converted value.
- **cache_ref_return** returns a conversion cache id.

The **XtCallConverter** function looks up the specified type converter in the application context associated with the display and, if the converter was not registered or was registered with cache type **XtCacheAll** or **XtCacheByDisplay**, looks in the conversion cache to see if this conversion procedure has been called with the specified conversion arguments. If so, it checks the success status of the prior call, and if the conversion failed, **XtCallConverter** returns **False** immediately; otherwise it checks the size specified in the **to** argument and, if it is greater than or equal to the size stored in the cache, copies the information stored in the cache into the location specified by **to->addr**, stores the cache size into **to->size**, and returns **True**. If the size specified in the **to** argument is smaller than the size stored in the cache, **XtCallConverter** copies the cache size into **to->size** and returns **False**. If the converter was registered with cache type **XtCacheNone** or no value was found in the conversion cache, **XtCallConverter** calls the converter and, if it was not registered with cache type **XtCacheNone**, enters the result in the cache. **XtCallConverter** then returns what the converter returned.

The **cache_ref_return** field specifies storage allocated by the caller in which an opaque value will be stored. If the type converter has been registered with the **XtCacheRefCount** modifier and if the value returned in **cache_ref_return** is non-NULL, then the caller should store the **cache_ref_return** value in order to decrement the reference count when the converted value is no longer required. The **cache_ref_return** argument should be NULL if the caller is unwilling or unable to store the value.

To explicitly decrement the reference counts for resources obtained from **XtCallConverter**, use **XtAppReleaseCacheRefs**.

```c
void XtAppReleaseCacheRefs(app_context, refs)
    XtApplicationContext app_context;
    XtCacheRef *refs;
```

- **app_context** specifies the application context.
- **refs** specifies the list of cache references to be released.

**XtAppReleaseCacheRefs** decrements the reference count for the conversion entries identified by the **refs** argument. This argument is a pointer to a NULL-terminated list of **XtCacheRef** values. If any reference count reaches zero, the destructor, if any, will be called and the resource removed from the conversion cache.
As a convenience to clients needing to explicitly decrement reference counts via a callback function, the Intrinsics define two callback procedures, `XtCallbackReleaseCacheRef` and `XtCallbackReleaseCacheRefList`.

```c
void XtCallbackReleaseCacheRef(object, client_data, call_data)
    Widget object;
    XtPointer client_data;
    XtPointer call_data;

    object Specifies the object with which the resource is associated.
    client_data Specifies the conversion cache entry to be released.
    call_data Is ignored.
```

This callback procedure may be added to a callback list to release a previously returned `XtCacheRef` value. When adding the callback, the callback `client_data` argument must be specified as the value of the `XtCacheRef` data cast to type `XtPointer`.

```c
void XtCallbackReleaseCacheRefList(object, client_data, call_data)
    Widget object;
    XtPointer client_data;
    XtPointer call_data;

    object Specifies the object with which the resources are associated.
    client_data Specifies the conversion cache entries to be released.
    call_data Is ignored.
```

This callback procedure may be added to a callback list to release a list of previously returned `XtCacheRef` values. When adding the callback, the callback `client_data` argument must be specified as a pointer to a NULL-terminated list of `XtCacheRef` values.

To lookup and call a resource converter, copy the resulting value, and free a cached resource when a widget is destroyed, use `XtConvertAndStore`.

```c
Boolean XtConvertAndStore(object, from_type, from, to_type, to_in_out)
    Widget object;
    String from_type;
    XmValuePtr from;
    String to_type;
    XmValuePtr to_in_out;

    object Specifies the object to use for additional arguments, if any are needed, and the destroy callback list. Must be of class Object or any subclass thereof.
    from_type Specifies the source type.
    from Specifies the value to be converted.
    to_type Specifies the destination type.
    to_in_out Specifies a descriptor for storage into which the converted value will be returned.
```

The `XtConvertAndStore` function looks up the type converter registered to convert `from_type` to `to_type`, computes any additional arguments needed, and then calls `XtCallConverter` (or `XtDirectConvert` if an old-style converter was registered with `XtAddConverter` or `XtAppAddConverter`; see Appendix C) with the `from` and `to_in_out` arguments. The `to_in_out` argument specifies the size and location into which the converted value will be stored and is passed directly to the converter. If the location is specified as NULL, it will be replaced with a pointer to private storage and the size will be returned in the descriptor. The caller is expected to copy this private storage immediately and must not modify it in any way. If a
non-NULL location is specified, the caller must allocate sufficient storage to hold the converted value and must also specify the size of that storage in the descriptor. The size field will be modified on return to indicate the actual size of the converted data. If the conversion succeeds, XtConvertAndStore returns True; otherwise, it returns False.

XtConvertAndStore adds XtCallbackReleaseCacheRef to the destroyCallback list of the specified object if the conversion returns an XtCacheRef value. The resulting resource should not be referenced after the object has been destroyed.

XtCreateWidget performs processing equivalent to XtConvertAndStore when initializing the object instance. Because there is extra memory overhead required to implement reference counting, clients may distinguish those objects that are never destroyed before the application exits from those that may be destroyed and whose resources should be deallocated.

To specify whether reference counting is to be enabled for the resources of a particular object when the object is created, the client can specify a value for the Boolean resource XtNinitialResourcesPersistent, class XtCInitialResourcesPersistent. When XtCreateWidget is called, if this resource is not specified as False in either the arglist or the resource database, then the resources referenced by this object are not reference-counted, regardless of how the type converter may have been registered. The effective default value is True; thus clients that expect to destroy one or more objects and want resources deallocated must explicitly specify False for XtNinitialResourcesPersistent.

The resources are still freed and destructors called when XtCloseDisplay is called if the conversion was registered as XtCacheByDisplay.

9.7. Reading and Writing Widget State

Any resource field in a widget can be read or written by a client. On a write operation, the widget decides what changes it will actually allow and updates all derived fields appropriately.

9.7.1. Obtaining Widget State

To retrieve the current values of resources associated with a widget instance, use XtGetValues.

```c
void XtGetValues(object, args, num_args)
    Widget object;
    ArgList args;
    Cardinal num_args;

object Specifies the object whose resource values are to be returned. Must be of class Object or any subclass thereof.

args Specifies the argument list of name/address pairs that contain the resource names and the addresses into which the resource values are to be stored. The resource names are widget-dependent.

num_args Specifies the number of entries in the argument list.
```

The XtGetValues function starts with the resources specified for the Object class and proceeds down the subclass chain to the class of the object. The value field of a passed argument list must contain the address into which to copy the contents of the corresponding object instance field. If the field is a pointer type, the lifetime of the pointed-to data is defined by the object class. For the Intrinsics-defined resources, the following lifetimes apply

- Not valid following any operation that modifies the resource:
  - XtNchildren resource of composite widgets.
  - All resources of representation type XtRCallback.
• Remain valid at least until the widget is destroyed:
  - XtNaccelerators, XtNtranslations.
• Remain valid until the Display is closed:
  - XtNscreen.

It is the caller’s responsibility to allocate and deallocate storage for the copied data according to the size of the resource representation type used within the object.

If the class of the object’s parent is a subclass of constraintWidgetClass, XtGetValues then fetches the values for any constraint resources requested. It starts with the constraint resources specified for constraintWidgetClass and proceeds down the subclass chain to the parent’s constraint resources. If the argument list contains a resource name that is not found in any of the resource lists searched, the value at the corresponding address is not modified. If any get_values_hook procedures in the object’s class or superclass records are non-NULL, they are called in superclass-to-subclass order after all the resource values have been fetched by XtGetValues. Finally, if the object’s parent is a subclass of constraintWidgetClass, and if any of the parent’s class or superclass records have declared ConstraintClassExtension records in the Constraint class part extension field with a record type of NULLQUARK and if the get_values_hook field in the extension record is non-NULL, XtGetValues calls the get_values_hook procedures in superclass-to-subclass order. This permits a Constraint parent to provide nonresource data via XtGetValues.

Get_values_hook procedures may modify the data stored at the location addressed by the value field, including (but not limited to) making a copy of data whose resource representation is a pointer. None of the Intrinsics-defined object classes copy data in this manner. Any operation that modifies the queried object resource may invalidate the pointed-to data.

To retrieve the current values of resources associated with a widget instance using varargs lists, use XtVaGetValues.

```c
void XtVaGetValues(object, ...)

Widget object;

object Specifies the object whose resource values are to be returned. Must be of class
Object or any subclass thereof.
...

void XtVaGetValues(Widget, ArgList, Cardinal*);

NumArg ArgList;
NumArg *NumArg;
```

XtVaGetValues is identical in function to XtGetValues with the args and num_args parameters replaced by a varargs list, as described in Section 2.5.1. All value entries in the list must specify pointers to storage allocated by the caller to which the resource value will be copied. It is the caller’s responsibility to ensure that sufficient storage is allocated. If XtVaTypedArg is specified, the type argument specifies the representation desired by the caller and the size argument specifies the number of bytes allocated to store the result of the conversion. If the size is insufficient, a warning message is issued and the list entry is skipped.

### 9.7.1.1. Widget Subpart Resource Data: the get_values_hook Procedure

Widgets that have subparts can return resource values from them through XtGetValues by supplying a get_values_hook procedure. The get_values_hook procedure pointer is of type XtArgsProc.

```c
typedef void (*XtArgsProc)(Widget, ArgList, Cardinal*);

Widget w;
ArgList args;
Cardinal *NumArg;
```
Specifies the widget whose subpart resource values are to be retrieved.

Specifies the argument list that was passed to XtGetValues or the transformed
varargs list passed to XtVaGetValues.

Specifies the number of entries in the argument list.

The widget with subpart resources should call XtGetSubvalues in the get_values_hook pro-
cedure and pass in its subresource list and the args and num_args parameters.

### 9.7.1.2. Widget Subpart State

To retrieve the current values of subpart resource data associated with a widget instance, use
XtGetSubvalues. For a discussion of subpart resources, see Section 9.4.

```c
void XtGetSubvalues(base, resources, num_resources, args, num_args)
    XtPointer base;
    XtResourceList resources;
    Cardinal num_resources;
    ArgList args;
    Cardinal num_args;
```

- **base** Specifies the base address of the subpart data structure for which the resources
  should be retrieved.
- **resources** Specifies the subpart resource list.
- **num_resources** Specifies the number of entries in the resource list.
- **args** Specifies the argument list of name/address pairs that contain the resource
  names and the addresses into which the resource values are to be stored.
- **num_args** Specifies the number of entries in the argument list.

The XtGetSubvalues function obtains resource values from the structure identified by base.
The value field in each argument entry must contain the address into which to store the
Corresponding resource value. It is the caller’s responsibility to allocate and deallocate this
storage according to the size of the resource representation type used within the subpart. If the
argument list contains a resource name that is not found in the resource list, the value at the
corresponding address is not modified.

To retrieve the current values of subpart resources associated with a widget instance using
varargs lists, use XtVaGetSubvalues.

```c
void XtVaGetSubvalues(base, resources, num_resources, ...)  
    XtPointer base;
    XtResourceList resources;
    Cardinal num_resources;
```

- **base** Specifies the base address of the subpart data structure for which the resources
  should be retrieved.
- **resources** Specifies the subpart resource list.
- **num_resources** Specifies the number of entries in the resource list.
- **...** Specifies a variable argument list of name/address pairs that contain the
  resource names and the addresses into which the resource values are to be stored.

XtVaGetSubvalues is identical in function to XtGetSubvalues with the args and num_args
parameters replaced by a varargs list, as described in Section 2.5.1. XtVaTypedArg is not
supported for XtVaGetSubvalues. If XtVaTypedArg is specified in the list, a warning mes-
 sage is issued and the entry is then ignored.
9.7.2. Setting Widget State

To modify the current values of resources associated with a widget instance, use `XtSetValues`

```c
void XtSetValues(object, args, num_args)
    Widget object;
    ArgList args;
    Cardinal num_args;

object Specifies the object whose resources are to be modified. Must be of class
Object or any subclass thereof.

args Specifies the argument list of name/value pairs that contain the resources to be
modified and their new values.

num_args Specifies the number of entries in the argument list.
```

The `XtSetValues` function starts with the resources specified for the Object class fields and
proceeds down the subclass chain to the object. At each stage, it replaces the `object` resource
fields with any values specified in the argument list. `XtSetValues` then calls the `set_values`
procedure for the object in superclass-to-subclass order. If the object has any non-NULL
`set_values_hook` fields, these are called immediately after the corresponding `set_values` pro-
cedure. This procedure permits subclasses to set subpart data via `XtSetValues`.

If the class of the object's parent is a subclass of `constraintWidgetClass`, `XtSetValues` also
updates the object's constraints. It starts with the constraint resources specified for `con-
straintWidgetClass` and proceeds down the subclass chain to the parent's class. At each
stage, it replaces the constraint resource fields with any values specified in the argument list. It
then calls the constraint `set_values` procedures from `constraintWidgetClass` down to the
parent's class. The constraint `set_values` procedures are called with widget arguments, as for
all `set_values` procedures, not just the constraint records, so that they can make adjustments to
the desired values based on full information about the widget. Any arguments specified that
do not match a resource list entry are silently ignored.

If the object is of a subclass of `RectObj`, `XtSetValues` determines if a geometry request is
needed by comparing the old object to the new object. If any geometry changes are required,
`XtSetValues` restores the original geometry and makes the request on behalf of the widget.a If
the geometry manager returns `XtGeometryYes`, `XtSetValues` calls the object's `resize` pro-
cedure. If the geometry manager returns `XtGeometryDone`, `XtSetValues` continues, as the
object's `resize` procedure should have been called by the geometry manager. If the geometry
manager returns `XtGeometryNo`, `XtSetValues` ignores the geometry request and continues.
If the geometry manager returns `XtGeometryAlmost`, `XtSetValues` calls the
`set_values_almost` procedure, which determines what should be done. `XtSetValues` then
repeats this process, deciding once more whether the geometry manager should be called.

Finally, if any of the `set_values` procedures returned `True`, and the widget is realized, `XtSet-
Values` causes the widget's `expose` procedure to be invoked by calling `XClearArea` on the
widget's window.

To modify the current values of resources associated with a widget instance using varargs lists,
use `XtVaSetValues`

```c
void XtVaSetValues(object, ...) 
    Widget object;

object Specifies the object whose resources are to be modified. Must be of class
Object or any subclass thereof.

... Specifies the variable argument list of name/value pairs that contain the
resources to be modified and their new values.
```

`XtVaSetValues` is identical in function to `XtSetValues` with the `args` and `num_args` param-
ters replaced by a varargs list, as described in Section 2.5.1.
9.7.2.1. Widget State: the set_values Procedure

The set_values procedure pointer in a widget class is of type XtSetValuesFunc.

```c
typedef Boolean (*XtSetValuesFunc)(Widget, Widget, Widget, ArgList, Cardinal*);
```

- `current` Specifies a copy of the widget as it was before the XtSetValues call.
- `request` Specifies a copy of the widget with all values changed as asked for by the XtSetValues call before any class set_values procedures have been called.
- `new` Specifies the widget with the new values that are actually allowed.
- `args` Specifies the argument list passed to XtSetValues or the transformed argument list passed to XtVaSetValues.
- `num_args` Specifies the number of entries in the argument list.

The set_values procedure should recompute any field derived from resources that are changed (for example, many GCs depend on foreground and background pixels). If no recomputation is necessary and if none of the resources specific to a subclass require the window to be redisplayed when their values are changed, you can specify NULL for the set_values field in the class record.

Like the initialize procedure, set_values mostly deals only with the fields defined in the subclass, but it has to resolve conflicts with its superclass, especially conflicts over width and height.

Sometimes a subclass may want to overwrite values filled in by its superclass. In particular, size calculations of a superclass are often incorrect for a subclass and, in this case, the subclass must modify or recalculate fields declared and computed by its superclass.

As an example, a subclass can visually surround its superclass display. In this case, the width and height calculated by the superclass set_values procedure are too small and need to be incremented by the size of the surround. The subclass needs to know if its superclass's size was calculated by the superclass or was specified explicitly. All widgets must place themselves into whatever size is explicitly given, but they should compute a reasonable size if no size is requested. How does a subclass know the difference between a specified size and a size computed by a superclass?

The request and new parameters provide the necessary information. The request widget is a copy of the widget, updated as originally requested. The new widget starts with the values in the request, but it has additionally been updated by all superclass set_values procedures called so far. A subclass set_values procedure can compare these two to resolve any potential conflicts. The set_values procedure need not refer to the request widget unless it must resolve conflicts between the current and new widgets. Any changes the widget needs to make, including geometry changes, should be made in the new widget.

In the above example, the subclass with the visual surround can see if the width and height in the request widget are zero. If so, it adds its surround size to the width and height fields in the new widget. If not, it must make do with the size originally specified. In this case, zero is a special value defined by the class to permit the application to invoke this behavior.

The new widget is the actual widget instance record. Therefore, the set_values procedure should do all its work on the new widget; the request widget should never be modified. If the set_values procedure needs to call any routines that operate on a widget, it should specify new as the widget instance.

Before calling the set_values procedures, the Intrinsics modify the resources of the request widget according to the contents of the arglist; if the widget names all its resources in the class
resource list, it is never necessary to examine the contents of args.

Finally, the set_values procedure must return a Boolean that indicates whether the widget needs to be redisplayed. Note that a change in the geometry fields alone does not require the set_values procedure to return True; the X server will eventually generate an Expose event, if necessary. After calling all the set_values procedures, XtSetValues forces a redisplay by calling XClearArea if any of the set_values procedures returned True. Therefore, a set_values procedure should not try to do its own redisplaying.

Set_values procedures should not do any work in response to changes in geometry because XtSetValues eventually will perform a geometry request, and that request might be denied. If the widget actually changes size in response to a call to XtSetValues, its resize procedure is called. Widgets should do any geometry-related work in their resize procedure.

Note that it is permissible to call XtSetValues before a widget is realized. Therefore, the set_values procedure must not assume that the widget is realized.

9.7.2.2. Widget State: the set_values_almost Procedure

The set_values_almost procedure pointer in the widget class record is of type XtAlmostProc.

define void (*XtAlmostProc)(Widget, Widget, XtWidgetGeometry*, XtWidgetGeometry*);

old
new
request
reply

old Specifies a copy of the object as it was before the XtSetValues call.
new Specifies the object instance record.
request Specifies the original geometry request that was sent to the geometry manager that caused XtGeometryAlmost to be returned.
reply Specifies the compromise geometry that was returned by the geometry manager with XtGeometryAlmost.

Most classes inherit the set_values_almost procedure from their superclass by specifying XtInheritSetValuesAlmost in the class initialization. The set_values_almost procedure in rectObjClass accepts the compromise suggested.

The set_values_almost procedure is called when a client tries to set a widget’s geometry by means of a call to XtSetValues, and the geometry manager cannot satisfy the request but instead returns XtGeometryNo or XtGeometryAlmost and a compromise geometry. The new object is the actual instance record. The x, y, width, height, and border_width fields contain the original values as they were before the XtSetValues call and all other fields contain the new values. The request parameter contains the new geometry request that was made to the parent. The reply parameter contains reply->request_mode equal to zero if the parent returned XtGeometryNo and contains the parent’s compromise geometry otherwise. The set_values_almost procedure takes the original geometry and the compromise geometry and determines if the compromise is acceptable or whether to try a different compromise. It returns its results in the request parameter, which is then sent back to the geometry manager for another try. To accept the compromise, the procedure must copy the contents of the reply geometry into the request geometry; to attempt an alternative geometry, the procedure may modify any part of the request argument; to terminate the geometry negotiation and retain the original geometry, the procedure must set request->request_mode to zero. The geometry fields of the old and new instances must not be modified directly.
9.7.2.3. Widget State: the ConstraintClassPart set_values Procedure

The constraint set_values procedure pointer is of type XtSetValuesFunc. The values passed to the parent’s constraint set_values procedure are the same as those passed to the child’s class set_values procedure. A class can specify NULL for the set_values field of the ConstraintPart if it need not compute anything.

The constraint set_values procedure should recompute any constraint fields derived from constraint resources that are changed. Further, it may modify other widget fields as appropriate. For example, if a constraint for the maximum height of a widget is changed to a value smaller than the widget’s current height, the constraint set_values procedure may reset the height field in the widget.

9.7.2.4. Widget Subpart State

To set the current values of subpart resources associated with a widget instance, use XtSetSubvalues. For a discussion of subpart resources, see Section 9.4.

```c
void XtSetSubvalues(base, resources, num_resources, args, num_args)
    XtPointer base;
    XtResourceList resources;
    Cardinal num_resources;
    ArgList args;
    Cardinal num_args;

    base Specifies the base address of the subpart data structure into which the resources should be written.
    resources Specifies the subpart resource list.
    num_resources Specifies the number of entries in the resource list.
    args Specifies the argument list of name/value pairs that contain the resources to be modified and their new values.
    num_args Specifies the number of entries in the argument list.
```

The XtSetSubvalues function updates the resource fields of the structure identified by base. Any specified arguments that do not match an entry in the resource list are silently ignored.

To set the current values of subpart resources associated with a widget instance using varargs lists, use XtVaSetSubvalues.

```c
void XtVaSetSubvalues(base, resources, num_resources, ...)     
    XtPointer base;
    XtResourceList resources;
    Cardinal num_resources;

    base Specifies the base address of the subpart data structure into which the resources should be written.
    resources Specifies the subpart resource list.
    num_resources Specifies the number of entries in the resource list.
    ... Specifies the variable argument list of name/value pairs that contain the resources to be modified and their new values.
```

XtVaSetSubvalues is identical in function to XtSetSubvalues with the args and num_args parameters replaced by a varargs list, as described in Section 2.5.1. XtVaTypedArg is not supported for XtVaSetSubvalues. If an entry containing XtVaTypedArg is specified in the list, a warning message is issued and the entry is ignored.
9.7.2.5. Widget Subpart Resource Data: the set_values_hook Procedure

Note

The set_values_hook procedure is obsolete, as the same information is now available to the set_values procedure. The procedure has been retained for those widgets that used it in versions prior to Release 4.

Widgets that have a subpart can set the subpart resource values through XtSetValues by supplying a set_values_hook procedure. The set_values_hook procedure pointer in a widget class is of type XtArgsFunc.

typedef Boolean (*XtArgsFunc)(Widget, Arglist, Cardinal*);

    Widget w;
    Arglist args;
    Cardinal *num_args;

    w Specifies the widget whose subpart resource values are to be changed.
    args Specifies the argument list that was passed to XtSetValues or the transformed
           varargs list passed to XtVaSetValues.
    num_args Specifies the number of entries in the argument list.

The widget with subpart resources may call XtSetValues from the set_values_hook procedure and pass in its subresource list and the args and num_args parameters.
Chapter 10

Translation Management

Except under unusual circumstances, widgets do not hardwire the mapping of user events into widget behavior by using the event manager. Instead, they provide a default mapping of events into behavior that you can override.

The translation manager provides an interface to specify and manage the mapping of X event sequences into widget-supplied functionality, for example, calling procedure *Abe* when the *y* key is pressed.

The translation manager uses two kinds of tables to perform translations:

- The action tables, which are in the widget class structure, specify the mapping of externally available procedure name strings to the corresponding procedure implemented by the widget class.
- A translation table, which is in the widget class structure, specifies the mapping of event sequences to procedure name strings.

You can override the translation table in the class structure for a specific widget instance by supplying a different translation table for the widget instance. The resources XtNtranslations and XtNbaseTranslations are used to modify the class default translation table; see Section 10.3.

10.1. Action Tables

All widget class records contain an action table, an array of XtActionsRec entries. In addition, an application can register its own action tables with the translation manager so that the translation tables it provides to widget instances can access application functionality directly.

The translation action procedure pointer is of type XtActionProc.

\[
\text{typedef void (*XtActionProc)(Widget, XEvent*, String*, Cardinal*)};
\]

\[
\text{w} \\
\text{event} \\
\text{params} \\
\text{num_params}
\]

Specifies the widget that caused the action to be called.

Specifies the event that caused the action to be called. If the action is called after a sequence of events, then the last event in the sequence is used.

Specifies a pointer to the list of strings that were specified in the translation table as arguments to the action, or NULL.

Specifies the number of entries in params.

typedef struct _XtActionsRec {
    String string;
    XtActionProc proc;
} XtActionsRec, *XtActionList;

The string field is the name used in translation tables to access the procedure. The proc field is a pointer to a procedure that implements the functionality.

When the action list is specified as the CoreClassPart actions field the string pointed to by string must be permanently allocated prior to or during the execution of the class initialization
procedure and must not be subsequently deallocated.

Action procedures should not assume that the widget in which they are invoked is realized; an accelerator specification can cause an action procedure to be called for a widget that does not yet have a window. Widget writers should also note which of a widget's callback lists are invoked from action procedures and warn clients not to assume the widget is realized in those callbacks.

For example, a Pushbutton widget has procedures to take the following actions:

- Set the button to indicate it is activated.
- Unset the button back to its normal mode.
- Highlight the button borders.
- Unhighlight the button borders.
- Notify any callbacks that the button has been activated.

The action table for the Pushbutton widget class makes these functions available to translation tables written for Pushbutton or any subclass. The string entry is the name used in translation tables. The procedure entry (usually spelled identically to the string) is the name of the C procedure that implements that function:

```c
XtActionsRec actionTable[] = {
    "Set", Set,
    "Unset", Unset,
    "Highlight", Highlight,
    "Unhighlight", Unhighlight,
    "Notify", Notify,
};
```

The Intrinsic reserve all action names and parameters starting with the characters "Xt" for future standard enhancements. Users, applications, and widgets should not declare action names or pass parameters starting with these characters except to invoke specified built-in Intrinsic functions.

10.1.1. Action Table Registration

The actions and num actions fields of CoreClassPart specify the actions implemented by a widget class. These are automatically registered with the Intrinsic when the class is initialized and must be allocated in writable storage prior to Core class_part initialization, and never deallocated. To save memory and optimize access, the Intrinsic may overwrite the storage in order to compile the list into an internal representation.

To declare an action table within an application and register it with the translation manager, use XtAppAddActions.

```c
void XtAppAddActions(app_context, actions, num_actions)
    XtAppContext app_context;
    XtActionList actions;
    Cardinal num_actions;

app_context Specifies the application context.
actions Specifies the action table to register.
num_actions Specifies the number of entries in this action table.
```
If more than one action is registered with the same name, the most recently registered action is used. If duplicate actions exist in an action table, the first is used. The Intrinsics register an action table containing XtMenuPopup and XtMenuPopdown as part of XtCreateApplicationContext.

10.1.2. Action Names to Procedure Translations
The translation manager uses a simple algorithm to resolve the name of a procedure specified in a translation table into the actual procedure specified in an action table. When the widget is realized, the translation manager performs a search for the name in the following tables, in order:

- The widget’s class and all superclass action tables, in subclass-to-superclass order.
- The parent’s class and all superclass action tables, in subclass-to-superclass order, then on up the ancestor tree.
- The action tables registered with XtAppAddActions and XtAddActions from the most recently added table to the oldest table.

As soon as it finds a name, the translation manager stops the search. If it cannot find a name, the translation manager generates a warning message.

10.1.3. Action Hook Registration
An application can specify a procedure that will be called just before every action routine is dispatched by the translation manager. To do so, the application supplies a procedure pointer of type XtActionHookProc.

typedef void (*XtActionHookProc)(Widget, XtPointer, String, XEvent*, String*, Cardinal*);

Widget w;
XtPointer client_data;
String action_name;
XEvent* event;
String* params;
Cardinal* num_params;

w Specifies the widget whose action is about to be dispatched.
client_data Specifies the application-specific closure that was passed to XtAppAddActionHook.
action_name Specifies the name of the action to be dispatched.
event Specifies the event argument that will be passed to the action routine.
params Specifies the action parameters that will be passed to the action routine.
um_params Specifies the number of entries in params.

Action hooks should not modify any of the data pointed to by the arguments other than the client_data argument.

To add an action hook, use XtAppAddActionHook.

XtActionHookId XtAppAddActionHook(app, proc, client_data)
    XtAppContext app;
    XtActionHookProc proc;
    XtPointer client_data;

app Specifies the application context.
proc Specifies the action hook procedure.
client_data Specifies application-specific data to be passed to the action hook.

XtAppAddActionHook adds the specified procedure to the front of a list maintained in the application context. In the future, when an action routine is about to be invoked for any widget in this application context, either through the translation manager or via XtCallActionProc, the action hook procedures will be called in reverse order of registration just prior to invoking the action routine.

Action hook procedures are removed automatically and the XtActionHookIds destroyed when the application context in which they were added is destroyed.

To remove an action hook procedure without destroying the application context, use XtRemoveActionHook.

```c
void XtRemoveActionHook(XtActionHookId id);
```

*id* Specifies the action hook id returned by XtAppAddActionHook.

XtRemoveActionHook removes the specified action hook procedure from the list in which it was registered.

### 10.2. Translation Tables

All widget instance records contain a translation table, which is a resource with a default value specified elsewhere in the class record. A translation table specifies what action procedures are invoked for an event or a sequence of events. A translation table is a string containing a list of translations from an event sequence into one or more action procedure calls. The translations are separated from one another by newline characters (ASCII LF). The complete syntax of translation tables is specified in Appendix B.

As an example, the default behavior of Pushbutton is

- Highlight on enter window.
- Unhighlight on exit window.
- Invert on left button down.
- Call callbacks and reinvert on left button up.

The following illustrates Pushbutton's default translation table:

```java
static String defaultTranslations =
"<EnterWindow>:Highlight()
<LeaveWindow>:Unhighlight()
 Btn1Down>: Set()
 Bnt1Up>: Notify() Unset();
```

The tm_table field of the CoreClassPart should be filled in at class initialization time with the string containing the class's default translations. If a class wants to inherit its superclass's translations, it can store the special value XtInheritTranslations into tm_table. In Core's class part initialization procedure, the Intrinsics compile this translation table into an efficient internal form. Then, at widget creation time, this default translation table is combined with the XtNtranslations and XtNbaseTranslations resources; see Section 10.3.

The resource conversion mechanism automatically compiles string translation tables that are specified in the resource database. If a client uses translation tables that are not retrieved via a resource conversion, it must compile them itself using XtParseTranslationTable.

The Intrinsics use the compiled form of the translation table to register the necessary events with the event manager. Widgets need do nothing other than specify the action and translation tables for events to be processed by the translation manager.
10.2.1. Event Sequences

An event sequence is a comma-separated list of X event descriptions that describes a specific sequence of X events to map to a set of program actions. Each X event description consists of three parts: The X event type, a prefix consisting of the X modifier bits, and an event-specific suffix.

Various abbreviations are supported to make translation tables easier to read. The events must match incoming events in left-to-right order to trigger the action sequence.

10.2.2. Action Sequences

Action sequences specify what program or widget actions to take in response to incoming X events. An action sequence consists of space-separated action procedure call specifications. Each action procedure call consists of the name of an action procedure and a parenthesized list of zero or more comma-separated string parameters to pass to that procedure. The actions are invoked in left-to-right order as specified in the action sequence.

10.2.3. Multi-click Time

Translation table entries may specify actions that are taken when two or more identical events occur consecutively within a short time interval, called the multi-click time. The multi-click time value may be specified as an application resource with name "multiClickTime" and class "MultiClickTime" and may also be modified dynamically by the application. The multi-click time is unique for each Display value and is retrieved from the resource database by XtDisplayInitialize. If no value is specified, the initial value is 200 milliseconds.

To set the multi-click time dynamically, use XtSetMultiClickTime.

```c
void XtSetMultiClickTime(Display *display, int time);
```

`display` Specifies the display connection.

`time` Specifies the multi-click time in milliseconds.

XtSetMultiClickTime sets the time interval used by the translation manager to determine when multiple events are interpreted as a repeated event. When a repeat count is specified in a translation entry, the interval between the timestamps in each pair of repeated events (e.g., between two ButtonPress events) must be less than the multi-click time in order for the translation actions to be taken.

To read the multi-click time, use XtGetMultiClickTime.

```c
int XtGetMultiClickTime(Display *display);
```

`display` Specifies the display connection.

XtGetMultiClickTime returns the time in milliseconds that the translation manager uses to determine if multiple events are to be interpreted as a repeated event for purposes of matching a translation entry containing a repeat count.

10.3. Translation Table Management

Sometimes an application needs to merge its own translations with a widget's translations. For example, a window manager provides functions to move a window. The window manager wishes to bind this operation to a specific pointer button in the title bar without the possibility
of user override and bind it to other buttons that may be overridden by the user.

To accomplish this, the window manager should first create the title bar and then should merge the two translation tables into the title bar’s translations. One translation table contains the translations that the window manager wants only if the user has not specified a translation for a particular event or event sequence (i.e., those that may be overridden). The other translation table contains the translations that the window manager wants regardless of what the user has specified.

Three Intrinsics functions support this merging:

- **XtParseTranslationTable**
  - Compiles a translation table.

- **XtAugmentTranslations**
  - Merges a compiled translation table into a widget’s compiled translation table, ignoring any new translations that conflict with existing translations.

- **XtOverrideTranslations**
  - Merges a compiled translation table into a widget’s compiled translation table, replacing any existing translations that conflict with new translations.

To compile a translation table, use **XtParseTranslationTable**.

```c
XtTranslations XtParseTranslationTable(String table);
```

- `table` Specifies the translation table to compile.

The **XtParseTranslationTable** function compiles the translation table, provided in the format given in Appendix B, into an opaque internal representation of type **XtTranslations**. Note that if an empty translation table is required for any purpose, one can be obtained by calling **XtParseTranslationTable** and passing an empty string.

To merge additional translations into an existing translation table, use **XtAugmentTranslations**.

```c
void XtAugmentTranslations(Widget w, XtTranslations translations);
```

- `w` Specifies the widget into which the new translations are to be merged. Must be of class Core or any subclass thereof.
- `translations` Specifies the compiled translation table to merge in.

The **XtAugmentTranslations** function merges the new translations into the existing widget translations, ignoring any #replace, #augment, or #override directive that may have been specified in the translation string. The translation table specified by `translations` is not altered by this process. **XtAugmentTranslations** logically appends the string representation of the new translations to the string representation of the widget’s current translations and reparses the result with no warning messages about duplicate left-hand sides, then stores the result back into the widget instance; i.e., if the new translations contain an event or event sequence that already exists in the widget’s translations, the new translation is ignored.

To overwrite existing translations with new translations, use **XtOverrideTranslations**.

```c
void XtOverrideTranslations(Widget w, XtTranslations translations);
```

- `w` Specifies the widget into which the new translations are to be merged. Must be of class Core or any subclass thereof.
- `translations` Specifies the compiled translation table to merge in.

The **XtOverrideTranslations** function merges the new translations into the existing widget translations, ignoring any #replace, #augment, or #override directive that may have been specified in the translation string. The translation table specified by `translations` is not altered by this process. **XtOverrideTranslations** logically appends the string representation of the new translations to the string representation of the widget’s current translations and reparses the result with no warning messages about duplicate left-hand sides, then stores the result back into the widget instance; i.e., if the new translations contain an event or event sequence that already exists in the widget’s translations, the new translation is ignored.
Specifies the widget into which the new translations are to be merged. Must be of class Core or any subclass thereof.

Specifies the compiled translation table to merge in.

The XtOverrideTranslations function merges the new translations into the existing widget translations, ignoring any replace, augment, or override directive that may have been specified in the translation string. The translation table specified by translations is not altered by this process. XtOverrideTranslations logically appends the string representation of the widget's current translations to the string representation of the new translations and reparses the result with no warning messages about duplicate left-hand sides, then stores the result back into the widget instance; i.e., if the new translations contain an event or event sequence that already exists in the widget's translations, the new translation overrides the widget's translation.

To replace a widget's translations completely, use XtSetValues on the XtNtranslations resource and specify a compiled translation table as the value.

To make it possible for users to easily modify translation tables in their resource files, the string-to-translation-table resource type converter allows the string to specify whether the table should replace, augment, or override any existing translation table in the widget. To specify this, a sharp sign (#) is given as the first character of the table followed by one of the keywords "replace", "augment", or "override" to indicate whether to replace, augment, or override the existing table. The replace or merge operation is performed during the Core instance initialization and during the Core set_values invocation. Each merge operation produces a new translation resource value; if the original tables were shared by other widgets, they are unaffected. If no directive is specified, "#replace" is assumed.

At instance initialization the XtNtranslations resource is first fetched. Then, if it was not specified or did not contain "#replace", the resource database is searched for the resource XtNbaseTranslations. If XtNbaseTranslations is found it is merged into the widget class translation table. Then the widget translations field is merged into the result, or into the class translation table if XtNbaseTranslations was not found. This final table is then stored into the widget translations field. If the XtNtranslations resource specified "#replace" no merge is done. If neither XtNbaseTranslations or XtNtranslations are specified, the class translation table is copied into the widget instance.

To completely remove existing translations, use XtUninstallTranslations.

void XtUninstallTranslations(w)
    Widget w;

w Specifies the widget from which the translations are to be removed. Must be of class Core or any subclass thereof.

The XtUninstallTranslations function causes the entire translation table for the widget to be removed.

10.4. Using Accelerators

It is often desirable to be able to bind events in one widget to actions in another. In particular, it is often useful to be able to invoke menu actions from the keyboard. The Intrinsics provide a facility, called accelerators, that lets you accomplish this. An accelerator table is a translation table that is bound with its actions in the context of a particular widget, the source widget. The accelerator table can then be installed on one or more destination widgets. When an event sequence in the destination widget would cause an accelerator action to be taken, and if the source widget is sensitive, the actions are executed as though triggered by the same event.
sequence in the accelerator source widget. The event is passed to the action procedure without modification. The action procedures used within accelerators must not assume that the source widget is realized nor that any fields of the event are in reference to the source widget's window if the widget is realized.

Each widget instance contains that widget's exported accelerator table as a resource. Each class of widget exports a method that takes a displayable string representation of the accelerators so that widgets can display their current accelerators. The representation is the accelerator table in canonical translation table form (see Appendix B). The display_accelerator procedure pointer is of type XtStringProc.

typedef void (*XtStringProc)(Widget, String);

void XtStringProc(Widget w, String string);

w Specifies the source widget that supplied the accelerators.

string Specifies the string representation of the accelerators for this widget.

Accelerators can be specified in resource files, and the string representation is the same as for a translation table. However, the interpretation of the #augment and #override directives applies to what will happen when the accelerator is installed; that is, whether or not the accelerator translations will override the translations in the destination widget. The default is #augment, which means that the accelerator translations have lower priority than the destination translations. The #replace directive is ignored for accelerator tables.

To parse an accelerator table, use XtParseAcceleratorTable.

XtAccelerators XtParseAcceleratorTable(source)

String source;

source Specifies the accelerator table to compile.

The XtParseAcceleratorTable function compiles the accelerator table into an opaque internal representation. The client should set the XtNaccelerators resource of each widget that is to be activated by these translations to the returned value.

To install accelerators from a widget on another widget, use XtInstallAccelerators.

void XtInstallAccelerators(destination, source)

Widget destination;

Widget source;

destination Specifies the widget on which the accelerators are to be installed. Must be of class Core or any subclass thereof.

source Specifies the widget from which the accelerators are to come. Must be of class Core or any subclass thereof.

The XtInstallAccelerators function installs the accelerators resource value from source onto destination by merging the the source accelerators into the destination translations. If the source display_accelerator field is non-NULL, XtInstallAccelerators calls it with the source widget and a string representation of the accelerator table, which indicates that its accelerators have been installed and that it should display them appropriately. The string representation of the accelerator table is its canonical translation table representation.

As a convenience for installing all accelerators from a widget and all its descendants onto one destination, use XtInstallAllAccelerators.
void XtInstallAllAccelerators(Widget destination; Widget source;)

destination  Specifies the widget on which the accelerators are to be installed. Must be of
class Core or any subclass thereof.

source     Specifies the root widget of the widget tree from which the accelerators are to
come. Must be of class Core or any subclass thereof.

The `XtInstallAllAccelerators` function recursively descends the widget tree rooted at `source`
and installs the accelerators resource value of each widget encountered onto `destination`. A
common use is to call `XtInstallAllAccelerators` and pass the application main window as the
source.

10.5. KeyCode-to-KeySym Conversions

The translation manager provides support for automatically translating KeyCodes in incoming
key events into KeySym. KeyCode-to-KeySym translator procedure pointers are of type
`XtKeyProc`.

typedef void (*XtKeyProc)(Display*, KeyCode, Modifiers, Modifiers*, KeySym*);

Display *display
KeyCode keycode
Modifiers modifiers
Modifiers *modifiers_return
KeySym *keysym_return

display Specifies the display that the KeyCode is from.

keycode Specifies the KeyCode to translate.

modifiers Specifies the modifiers to the KeyCode.

modifiers_return Specifies a location in which to store a mask that indicates the subset of all
modifiers that are examined by the key translator.

keysym_return Specifies a location in which to store the resulting KeySym.

This procedure takes a KeyCode and modifiers and produces a KeySym. For any given key
translator function, `modifiers_return` will be a constant that indicates the subset of all modifiers
that are examined by the key translator.

The KeyCode-to-KeySym translator procedure must be implemented such that multiple calls
with the same `display`, `keycode`, and `modifiers` return the same result until either a new case
converter ( `XtCaseProc`) is installed or a MappingNotify event is received.

The Intrinsics maintain tables internally to map KeyCodes to KeySym for each open display.
Translator procedures and other clients may share a single copy of this table to perform the
same mapping.

To return a pointer to the KeySym-to-KeyCode mapping table for a particular display, use
`XtGetKeysymTable`.

KeySym *XtGetKeysymTable(Display*, min_keycode, keysyms_per_keycode)

Display *display
KeyCode *min_keycode
int *keysyms_per_keycode

display Specifies the display whose table is required.

min_keycode Returns the minimum KeyCode valid for the display.
keysyms_per_keycode_return
Returns the number of KeySyms stored for each KeyCode.

XtGetKeysymTable returns a pointer to the Intrinsics' copy of the server's KeyCode-to-KeySym table. This table must not be modified. There are keysyms_per_keycode_return KeySyms associated with each KeyCode, located in the table with indices starting at index

(test_keycode - min_keycode_return) * keysyms_per_keycode_return
for KeyCode test_keycode. Any entries that have no KeySyms associated with them contain the value NoSymbol. Clients should not cache the KeySym table but should call XtGetKeysymTable each time the value is needed, as the table may change prior to dispatching each event.

For more information on this table, see Section 12.7 in Xlib – C Language X Interface.

To register a key translator, use XtSetKeyTranslator.

void XtSetKeyTranslator(Display* display, proc)
    Display *display;
    XtKeyProc proc;

display Specifies the display from which to translate the events.
proc Specifies the procedure to perform key translations.

The XtSetKeyTranslator function sets the specified procedure as the current key translator. The default translator is XtTranslateKey, an XtKeyProc that uses the Shift, Lock, and group modifiers with the interpretations defined in X Window System Protocol, Section 5. It is provided so that new translators can call it to get default KeyCode-to-KeySym translations and so that the default translator can be reinstalled.

To invoke the currently registered KeyCode-to-KeySym translator, use XtTranslateKeycode.

void XtTranslateKeycode(Display* display, KeyCode keycode, Modifiers *modifiers, Modifiers *modifiers_return, KeySym *keysym_return)
    Display *display;
    KeyCode keycode;
    Modifiers modifiers;
    Modifiers *modifiers_return;
    KeySym *keysym_return;

display Specifies the display that the KeyCode is from.
keycode Specifies the KeyCode to translate.
modifiers Specifies the modifiers to the KeyCode.
modifiers_return Returns a mask that indicates the modifiers actually used to generate the KeySym.
keysym_return Returns the resulting KeySym.

The XtTranslateKeycode function passes the specified arguments directly to the currently registered KeyCode-to-KeySym translator.

To handle capitalization of nonstandard KeySyms, the Intrinsics allow clients to register case conversion routines. Case converter procedure pointers are of type XtCaseProc.

typedef void (*XtCaseProc)(Display*, KeySym, KeySym*, KeySym*);
    Display *display;
    KeySym keysym;
    KeySym *lower_return;
    KeySym *upper_return;
display Specifies the display connection for which the conversion is required.
keysym Specifies the KeySym to convert.
lower_return Specifies a location into which to store the lower-case equivalent for the KeySym.
upper_return Specifies a location into which to store the upper-case equivalent for the KeySym.

If there is no case distinction, this procedure should store the KeySym into both return values.

To register a case converter, use XtRegisterCaseConverter.

```c
void XtRegisterCaseConverter(display, proc, start, stop)
  Display *display;
  XtCaseProc proc;
  KeySym start;
  KeySym stop;
```

display Specifies the display from which the key events are to come.
proc Specifies the XtCaseProc to do the conversions.
start Specifies the first KeySym for which this converter is valid.
stop Specifies the last KeySym for which this converter is valid.

The XtRegisterCaseConverter registers the specified case converter. The start and stop arguments provide the inclusive range of KeySyms for which this converter is to be called. The new converter overrides any previous converters for KeySyms in that range. No interface exists to remove converters; you need to register an identity converter. When a new converter is registered, the Intrinsics refresh the keyboard state if necessary. The default converter understands case conversion for all Latin KeySyms defined in X Window System Protocol, Appendix A.

To determine upper- and lower-case equivalents for a KeySym, use XtConvertCase.

```c
void XtConvertCase(display, keysym, lower_return, upper_return)
  Display *display;
  KeySym keysym;
  KeySym *lower_return;
  KeySym *upper_return;
```

display Specifies the display that the KeySym came from.
keysym Specifies the KeySym to convert.
lower_return Returns the lower-case equivalent of the KeySym.
upper_return Returns the upper-case equivalent of the KeySym.

The XtConvertCase function calls the appropriate converter and returns the results. A user-supplied XtKeyProc may need to use this function.

10.6. Obtaining a KeySym in an Action Procedure

When an action procedure is invoked on a KeyPress or KeyRelease event, it often has a need to retrieve the KeySym and modifiers corresponding to the event that caused it to be invoked. In order to avoid repeating the processing that was just performed by the Intrinsics to match the translation entry, the KeySym and modifiers are stored for the duration of the action procedure and are made available to the client.

To retrieve the KeySym and modifiers that matched the final event specification in the translation table entry, use XtGetActionKeysym.
KeySym XtGetActionKeysym(event, modifiers_return)
    XEvent *event;
    Modifiers *modifiers_return;

    event Specifies the event pointer passed to the action procedure by the Intrinsics.
    modifiers_return Returns the modifiers that caused the match, if non-NULL.

    If XtGetActionKeysym is called after an action procedure has been invoked by the Intrinsics
    and before that action procedure returns, and if the event pointer has the same value as the
    event pointer passed to that action routine, and if the event is a KeyPress or KeyRelease
    event, then XtGetActionKeysym returns the KeySym that matched the final event
    specification in the translation table and, if modifiers_return is non-NULL, the modifier state
    actually used to generate this KeySym; otherwise, if the event is a KeyPress or KeyRelease
    event, then XtGetActionKeysym calls XtTranslateKeycode and returns the results; else it
    returns NoSymbol and does not examine modifiers_return.

    Note that if an action procedure invoked by the Intrinsics invokes a subsequent action pro¬
    cedure (and so on) via XtCallActionProc, the nested action procedure may also call
    XtGetActionKeysym to retrieve the Intrinsics' KeySym and modifiers.

10.7. KeySym-to-KeyCode Conversions
To return the list of KeyCodes that map to a particular KeySym in the keyboard mapping table
maintained by the Intrinsics, use XtKeysymToKeycodeList.

void XtKeysymToKeycodeList(display, keysym, keycodes_return, keycount_return)
    Display *display;
    KeySym keysym;
    KeyCode **keycodes_return;
    Cardinal *keycount_return;

    display Specifies the display whose table is required.
    keysym Specifies the KeySym for which to search.
    keycodes_return Returns a list of KeyCodes that have keysym associated with them, or
    NULL if keycount_return is 0.
    keycount_return Returns the number of KeyCodes in the keycode list.

    The XtKeysymToKeycodeList procedure returns all the KeyCodes that have keysym in their
    entry for the keyboard mapping table associated with display. For each entry in the table, the
    first four KeySyms (groups 1 and 2) are interpreted as specified by X Window System Protocol,
    Section 5. If no KeyCodes map to the specified KeySym, keycount_return is zero and
    *keycodes_return is NULL.

    The caller should free the storage pointed to by keycodes_return using XtFree when it is no
    longer useful. If the caller needs to examine the KeyCode-to-KeySym table for a particular
    KeyCode, it should call XtGetKeysymTable.

10.8. Registering Button and Key Grabs For Actions
To register button and key grabs for a widget's window according to the event bindings in the
widget's translation table, use XtRegisterGrabAction.

void XtRegisterGrabAction(action_proc, owner_events, event_mask, pointer_mode, keyboard_mode)
    XtActionProc action_proc;
    Boolean owner_events;
    unsigned int event_mask;
    int pointer_mode, keyboard_mode;
action_proc  Specifies the action procedure to search for in translation tables.

owner_events  event_mask  pointer_mode
keyboard_mode  Specify arguments to XtGrabButton or XtGrabKey.

XtRegisterGrabAction  adds the specified action_proc to a list known to the translation manager. When a widget is realized, or when the translations of a realized widget or the accelerators installed on a realized widget are modified, its translation table and any installed accelerators are scanned for action procedures on this list. If any are invoked on ButtonPress or KeyPress events as the only or final event in a sequence, the Intrinsics will call XtGrabButton or XtGrabKey for the widget with every button or KeyCode which maps to the event detail field, passing the specified owner_events, event_mask, pointer_mode, and keyboard_mode. For ButtonPress events, the modifiers specified in the grab are determined directly from the translation specification and confine_to and cursor are specified as None. For KeyPress events, if the translation table entry specifies colon (:) in the modifier list, the modifiers are determined by calling the key translator procedure registered for the display and calling XtGrabKey for every combination of standard modifiers which map the KeyCode to the specified event detail KeySym, and ORing any modifiers specified in the translation table entry, and event_mask is ignored. If the translation table entry does not specify colon in the modifier list, the modifiers specified in the grab are those specified in the translation table entry only. For both ButtonPress and KeyPress events, don’t-care modifiers are ignored unless the translation entry explicitly specifies “Any” in the modifiers field.

If the specified action_proc is already registered for the calling process, the new values will replace the previously specified values for any widgets that become realized following the call, but existing grabs are not altered on currently-realized widgets.

When translations or installed accelerators are modified for a realized widget, any previous key or button grabs registered as a result of the old bindings are released if they do not appear in the new bindings and are not explicitly grabbed by the client with XtGrabKey or XtGrabButton.

10.9. Invoking Actions Directly

Normally action procedures are invoked by the Intrinsics when an event or event sequence arrives for a widget. To invoke an action procedure directly, without generating (or synthesizing) events, use XtCallActionProc.

void XtCallActionProc(widget, action, event, params, num_params)
  Widget widget;
  String action;
  XEvent *event;
  String *params;
  Cardinal num_params;

XtCallActionProc searches for the named action routine in the same manner and order as translation tables are bound, as described in Section 10.1.2, except that application action tables are searched, if necessary, as of the time of the call to XtCallActionProc. If found, the action
routine is invoked with the specified widget, event pointer, and parameters. It is the responsibility of the caller to ensure that the contents of the \textit{event}, \textit{params}, and \textit{num\_params} arguments are appropriate for the specified action routine and, if necessary, that the specified widget is realized or sensitive. If the named action routine cannot be found, \texttt{XtCallActionProc} generates a warning message and returns.

10.10. Obtaining a Widget's Action List

Occasionally a subclass will require the pointers to one or more of its superclass's action procedures. This would be needed, for example, in order to envelope the superclass's action. To retrieve the list of action procedures registered in the superclass's \textit{actions} field, use \texttt{XtGetActionList}.

\begin{verbatim}
void XtGetActionList(widget_class, actions_return, num_actions_return)
    WidgetClass widget_class;
    XtActionList *actions_return;
    Cardinal *num_actions_return;

    widget_class    Specifies the widget class whose actions are to be returned.
    actions_return  Returns the action list.
    num_actions_return
                        Returns the number of action procedures declared by the class.
\end{verbatim}

\texttt{XtGetActionList} returns the action table defined by the specified widget class. This table does not include actions defined by the superclasses. If \texttt{widget\_class} is not initialized, or is not \texttt{coreWidgetClass} or a subclass thereof, or if the class does not define any actions, \texttt{*actions\_return} will be NULL and \texttt{*num\_actions\_return} will be zero. If \texttt{*actions\_return} is non-NULL the client is responsible for freeing the table using \texttt{XtFree} when it is no longer needed.
Chapter 11
Utility Functions

The Intrinsics provide a number of utility functions that you can use to
 • Determine the number of elements in an array.
 • Translate strings to widget instances.
 • Manage memory usage.
 • Share graphics contexts.
 • Manipulate selections.
 • Merge exposure events into a region.
 • Translate widget coordinates.
 • Locate a widget given a window id.
 • Handle errors.
 • Set the WM_COLORMAP_WINDOWS property.
 • Locate files by name with string substitutions.

11.1. Determining the Number of Elements in an Array
To determine the number of elements in a fixed-size array, use XtNumber.
Cardinal XtNumber(array)
   ArrayType array;

array Specifies a fixed-size array of arbitrary type.
The XtNumber macro returns the number of elements allocated to the array.

11.2. Translating Strings to Widget Instances
To translate a widget name to a widget instance, use XtNameToWidget.
Widget XtNameToWidget(reference, names)
   Widget reference;
   String names;

reference Specifies the widget from which the search is to start. Must be of class Core
   or any subclass thereof.
names Specifies the partially qualified name of the desired widget.
The XtNameToWidget function searches for a descendant of the reference widget whose
name matches the specified names. The names parameter specifies a simple object name or a
series of simple object name components separated by periods or asterisks. XtNameToWidget
returns the descendant with the shortest name matching the specification according
to the following rules, where child is either a pop-up child or a normal child if the widget’s
class is a subclass of Composite :
 • Enumerate the object subtree rooted at the reference widget in breadth-first order, quali-
fying the name of each object with the names of all its ancestors up to but not including
the reference widget. The ordering between children of a common parent is not defined.
• Return the first object in the enumeration that matches the specified name, where each component of \texttt{names} matches exactly the corresponding component of the qualified object name, and asterisk matches any series of components, including none.

• If no match is found, return \texttt{NULL}.

Since breadth-first traversal is specified, the descendant with the shortest matching name (i.e., the fewest number of components), if any, will always be returned. However, since the order of enumeration of children is undefined and since the Intrinsics do not require that all children of a widget have unique names, \texttt{XtNameToWidget} may return any child that matches if there are multiple objects in the subtree with the same name. Consecutive separators (periods or asterisks) including at least one asterisk are treated as a single asterisk. Consecutive periods are treated as a single period.

11.3. Managing Memory Usage

The Intrinsics’ memory management functions provide uniform checking for null pointers and error reporting on memory allocation errors. These functions are completely compatible with their standard C language runtime counterparts \texttt{malloc}, \texttt{calloc}, \texttt{realloc}, and \texttt{free} with the following added functionality:

• \texttt{XtMalloc}, \texttt{XtCalloc}, and \texttt{XtRealloc} give an error if there is not enough memory.

• \texttt{XtFree} simply returns if passed a \texttt{NULL} pointer.

• \texttt{XtRealloc} simply allocates new storage if passed a \texttt{NULL} pointer.

See the standard C library documentation on \texttt{malloc}, \texttt{calloc}, \texttt{realloc}, and \texttt{free} for more information.

To allocate storage, use \texttt{XtMalloc}.

\begin{verbatim}
char *XtMalloc(size)
    Cardinal size;

size        Specifies the number of bytes desired.
\end{verbatim}

The \texttt{XtMalloc} function returns a pointer to a block of storage of at least the specified \texttt{size} bytes. If there is insufficient memory to allocate the new block, \texttt{XtMalloc} calls \texttt{XtErrorMsg}.

To allocate and initialize an array, use \texttt{XtCalloc}.

\begin{verbatim}
char *XtCalloc(num, size)
    Cardinal num;
    Cardinal size;

num        Specifies the number of array elements to allocate.
size        Specifies the size of each array element in bytes.
\end{verbatim}

The \texttt{XtCalloc} function allocates space for the specified number of array elements of the specified size and initializes the space to zero. If there is insufficient memory to allocate the new block, \texttt{XtCalloc} calls \texttt{XtErrorMsg}. \texttt{XtCalloc} returns the address of the allocated storage.

To change the size of an allocated block of storage, use \texttt{XtRealloc}.

\begin{verbatim}
char *XtRealloc(ptr, num)
    char *ptr;
    Cardinal num;

ptr        Specifies a pointer to the old storage allocated with \texttt{XtMalloc}, \texttt{XtCalloc}, or \texttt{XtRealloc}, or \texttt{NULL}.
\end{verbatim}
num Specifies number of bytes desired in new storage.
The XtRealloc function changes the size of a block of storage, possibly moving it. Then it copies the old contents (or as much as will fit) into the new block and frees the old block. If there is insufficient memory to allocate the new block, XtRealloc calls XtErrorMsg. If ptr is NULL, XtRealloc simply calls XtMalloc. XtRealloc then returns the address of the new block.

To free an allocated block of storage, use XtFree.

```c
void XtFree(ptr)
char *ptr;
```

ptr Specifies a pointer to a block of storage allocated with XtMalloc, XtCalloc, or XtRealloc, or NULL.
The XtFree function returns storage, allowing it to be reused. If ptr is NULL, XtFree returns immediately.

To allocate storage for a new instance of a type, use XtNew.

type *XtNew(type)
  type t;

type Specifies a previously declared type.
XtNew returns a pointer to the allocated storage. If there is insufficient memory to allocate the new block, XtNew calls XtErrorMsg. XtNew is a convenience macro that calls XtMalloc with the following arguments specified:

```c
((type *) XtMalloc((unsigned) sizeof(type)))
```
The storage allocated by XtNew should be freed using XtFree.

To copy an instance of a string, use XtNewString.

```c
String XtNewString(string)
  String string;
```

string Specifies a previously declared string.
XtNewString returns a pointer to the allocated storage. If there is insufficient memory to allocate the new block, XtNewString calls XtErrorMsg. XtNewString is a convenience macro that calls XtMalloc with the following arguments specified:

```c
strcpy(XtMalloc((unsigned)strlen(str) + 1), str))
```
The storage allocated by XtNewString should be freed using XtFree.

11.4. Sharing Graphics Contexts
The Intrinsics provide a mechanism whereby cooperating objects can share a graphics context (GC), thereby reducing both the number of GCs created and the total number of server calls in any given application. The mechanism is a simple caching scheme and allows for clients to declare both modifiable and nonmodifiable fields of the shared GCs.
To obtain a shareable GC with modifiable fields, use XtAllocateGC.
GC XtAllocateGC(widget, depth, value_mask, values, dynamic_mask, unused_mask)  
| Widget object; |
| Cardinal depth; |
| XtGCMask value_mask; |
| XGCValues *values; |
| XtGCMask dynamic_mask; |
| XtGCMask unused_mask; |

object Specifies an object, giving the screen for which the returned GC is valid. Must be of class Object or any subclass thereof.
depth Specifies the depth for which the returned GC is valid, or 0.
value_mask Specifies fields of the GC that are initialized from values.
values Specifies the values for the initialized fields.
dynamic_mask Specifies fields of the GC that may be modified by the caller.
unused_mask Specifies fields of the GC that will not be used by the caller.

The XtAllocateGC function returns a shareable GC that may be modified by the client. The screen field of the specified widget or of the nearest widget ancestor of the specified object and the specified depth argument supply the root and drawable depths for which the GC is to be valid. If depth is zero the depth is taken from the depth field of the specified widget or of the nearest widget ancestor of the specified object.

The value_mask argument specifies fields of the GC that will be initialized with the respective member of the values structure. The dynamic_mask argument specifies fields that the caller intends to modify during program execution. The caller must insure that the corresponding GC field is set prior to each use of the GC. The unused_mask argument specifies fields of the GC that are of no interest to the caller. The caller may assume that at all times all fields not specified in either dynamic_mask or unused_mask have their default value if not specified in value_mask or the value specified by values. If a field is specified in both value_mask and dynamic_mask, the effect is as if it were specified only in dynamic_mask and then immediately set to the value in values. If a field is set in unused_mask and also in either value_mask or dynamic_mask, the specification in unused_mask is ignored.

XtAllocateGC tries to minimize the number of unique GCs created by comparing the arguments with those of previous calls and returning an existing GC when there are no conflicts. XtAllocateGC may modify and return an existing GC if it was allocated with a nonzero unused_mask.

To obtain a shareable GC with no modifiable fields, use XtGetGC.

GC XtGetGC(object, value_mask, values)  
| Widget object; |
| XtGCMask value_mask; |
| XGCValues *values; |

object Specifies an object, giving the screen and depth for which the returned GC is valid. Must be of class Object or any subclass thereof.
value_mask Specifies which fields of the values structure are specified.
values Specifies the actual values for this GC.
The XtGetGC function returns a shareable, read-only GC. The parameters to this function are the same as those for XCreateGC except that an Object is passed instead of a Display. XtGetGC is equivalent to XtAllocateGC with depth, dynamic_mask, and unused_mask all zero.

XtGetGC shares only GCs in which all values in the GC returned by XCreateGC are the same. In particular, it does not use the value_mask provided to determine which fields of the GC a widget considers relevant. The value_mask is used only to tell the server which fields should be filled in from values and which it should fill in with default values.

To deallocate a shared GC when it is no longer needed, use XtReleaseGC.

void XtReleaseGC(object, gc)

    Widget object;
    GC gc;

object Specifies any object on the Display for which the GC was created. Must be of class Object or any subclass thereof.

gc Specifies the shared GC obtained with either XtAllocateGC or XtGetGC.

References to shareable GCs are counted and a free request is generated to the server when the last user of a given GC releases it.

11.5. Managing Selections

Arbitrary widgets in multiple applications can communicate with each other by means of the Intrinsics global selection mechanism, which conforms to the specifications in the Inter-Client Communication Conventions Manual. The Intrinsics supply functions for providing and receiving selection data in one logical piece (atomic transfers) or in smaller logical segments (incremental transfers).

The incremental interface is provided for a selection owner or selection requestor that cannot or prefers not to pass the selection value to and from the Intrinsics in a single call. For instance, either an application that is running on a machine with limited memory may not be able to store the entire selection value in memory, or a selection owner may already have the selection value available in discrete chunks, and it would be more efficient not to have to allocate additional storage to copy the pieces contiguously. Any owner or requestor that prefers to deal with the selection value in segments can use the incremental interfaces to do so. The transfer between the selection owner or requestor and the Intrinsics is not required to match the underlying transport protocol between the application and the X server; the Intrinsics will break a too large selection into smaller pieces for transport if necessary and will coalesce a selection transmitted incrementally if the value was requested atomically.

11.5.1. Setting and Getting the Selection Timeout Value

To set the Intrinsics selection timeout, use XtAppSetSelectionTimeout.

void XtAppSetSelectionTimeout(app_context, timeout)

    XtAppContext app_context;
    unsigned long timeout;

app_context Specifies the application context.

timeout Specifies the selection timeout in milliseconds.

To get the current selection timeout value, use XtAppGetSelectionTimeout.
unsigned long XtAppGetSelectionTimeout(app_context)
XtAppContext app_context;

app_context Specifies the application context.
The XtAppGetSelectionTimeout function returns the current selection timeout value, in milliseconds. The selection timeout is the time within which the two communicating applications must respond to one another. The initial timeout value is set by the selectionTimeout application resource as retrieved by XtDisplayInitialize. If selectionTimeout is not specified, the default is five seconds.

11.5.2. Using Atomic Transfers

When using atomic transfers, the owner will completely process one selection request at a time. The owner may consider each request individually, since there is no possibility for overlap between evaluation of two requests.

11.5.2.1. Atomic Transfer Procedures

The following procedures are used by the selection owner when providing selection data in a single unit.

The procedure pointer specified by the owner to supply the selection data to the Intrinsics is of type XtConvertSelectionProc.

XtPointer*, unsigned long*, int*);

Widget w;
Atom *selection;
Atom *target;
Atom *type_return;
XtPointer *value_return;
unsigned long *length_return;
int *format_return;

w Specifies the widget that currently owns this selection.
selection Specifies the atom naming the selection requested (for example, XA_PRIMARY or XA_SECONDARY).
target Specifies the target type of the selection that has been requested, which indicates the desired information about the selection (for example, File Name, Text, Window).
type_return Specifies a pointer to an atom into which the property type of the converted value of the selection is to be stored. For instance, either File Name or Text might have property type XA_STRING.
value_return Specifies a pointer into which a pointer to the converted value of the selection is to be stored. The selection owner is responsible for allocating this storage. If the selection owner has provided an XtSelectionDoneProc for the selection, this storage is owned by the selection owner; otherwise, it is owned by the Intrinsics selection mechanism, which frees it by calling XtFree when it is done with it.
length_return Specifies a pointer into which the number of elements in value_return, each of size indicated by format_return, is to be stored.
format_return Specifies a pointer into which the size in bits of the data elements of the selection value is to be stored.
This procedure is called by the Intrinsics selection mechanism to get the value of a selection as a given type from the current selection owner. It returns True if the owner successfully converted the selection to the target type or False otherwise. If the procedure returns False, the values of the return arguments are undefined. Each XtConvertSelectionProc should respond to target value TARGETS by returning a value containing the list of the targets into which it is prepared to convert the selection. The value returned in format_return must be one of 8, 16, or 32 to allow the server to byte-swap the data if necessary.

This procedure does not need to worry about responding to the MULTIPLE or the TIMES-TAMP target values (see Section 2.6.2 in the Inter-Client Communication Conventions Manual). A selection request with the MULTIPLE target type will be transparently transformed into a series of calls to this procedure, one for each target type, and a selection request with the TIMESTAMP target value will be answered automatically by the Intrinsics using the time specified in the call to XtOwnSelection or XtOwnSelectionIncremental.

To retrieve the SelectionRequest event that triggered the XtConvertSelectionProc procedure, use XtGetSelectionRequest.

XSelectionRequestEvent *XtGetSelectionRequest(w, selection, request_id)

 Widget w;
 Atom selection;
 XtRequestId request_id;

 w Specifies the widget that currently owns this selection. Must be of class Core or any subclass thereof.
 selection Specifies the selection being processed.
 request_id Specifies the requestor id in the case of incremental selections, or NULL in the case of atomic transfers.

XtGetSelectionRequest may only be called from within an XtConvertSelectionProc procedure and returns a pointer to the SelectionRequest event that caused the conversion procedure to be invoked. Request_id specifies a unique id for the individual request in the case that multiple incremental transfers are outstanding. For atomic transfers, request_id must be specified as NULL. If no SelectionRequest event is being processed for the specified widget, selection, and request_id, XtGetSelectionRequest returns NULL.

The procedure pointer specified by the owner when it desires notification upon losing ownership is of type XtLoseSelectionProc.

typedef void (*XtLoseSelectionProc)(Widget, Atom*);

 Widget w;
 Atom *selection;

 w Specifies the widget that has lost selection ownership.
 selection Specifies the atom naming the selection.

This procedure is called by the Intrinsics selection mechanism to inform the specified widget that it has lost the given selection. Note that this procedure does not ask the widget to relinquish the selection ownership; it is merely informative.

The procedure pointer specified by the owner when it desires notification of receipt of the data or when it manages the storage containing the data is of type XtSelectionDoneProc.
typedef void (*XtSelectionDoneProc)(Widget, Atom*, Atom*);
    Widget w;
    Atom *selection;
    Atom *target;

w Specifies the widget that owns the converted selection.

selection Specifies the atom naming the selection that was converted.

target Specifies the target type to which the conversion was done.

This procedure is called by the Intrinsics selection mechanism to inform the selection owner that a selection requestor has successfully retrieved a selection value. If the selection owner has registered an XtSelectionDoneProc, it should expect it to be called once for each conversion that it performs, after the converted value has been successfully transferred to the requestor. If the selection owner has registered an XtSelectionDoneProc, it also owns the storage containing the converted selection value.

11.5.2.2. Getting the Selection Value

The procedure pointer specified by the requestor to receive the selection data from the Intrinsics is of type XtSelectionCallbackProc.

typedef void (*XtSelectionCallbackProc)(Widget, XtPointer, Atom*, Atom*, XtPointer, unsigned long*, int);
    Widget w;
    XtPointer client_data;
    Atom *selection;
    Atom *type;
    XtPointer value;
    unsigned long *length;
    int *format;

w Specifies the widget that requested the selection value.

client_data Specifies a value passed in by the widget when it requested the selection.

selection Specifies the name of the selection that was requested.

type Specifies the representation type of the selection value (for example, XA_STRING). Note that it is not the target that was requested (which the client must remember for itself) but the type that is used to represent the target. The special symbolic constant XT_CONVERT_FAIL is used to indicate that the selection conversion failed because the selection owner did not respond within the Intrinsics selection timeout interval.

value Specifies a pointer to the selection value. The requesting client owns this storage and is responsible for freeing it by calling XtFree when it is done with it.

length Specifies the number of elements in value.

format Specifies the size in bits of the data elements of value.

This procedure is called by the Intrinsics selection mechanism to deliver the requested selection to the requestor.

If the SelectionNotify event returns a property of None, meaning the conversion has been refused because there is no owner for the specified selection or the owner cannot convert the selection to the requested target for any reason, the procedure is called with a value of NULL and a length of zero.

To obtain the selection value in a single logical unit, use XtGetSelectionValue or XtGetSelectionValues.
void XtGetSelectionValue(w, selection, target, callback, client_data, time)
  Widget w;
  Atom selection;
  Atom target;
  XtSelectionCallbackProc callback;
  XtPointer client_data;
  Time time;

w Specifies the widget making the request. Must be of class Core or any subclass thereof.

selection Specifies the particular selection desired; for example, XA_PRIMARY.
target Specifies the type of information needed about the selection.
callback Specifies the procedure to be called when the selection value has been obtained. Note that this is how the selection value is communicated back to the client.

client_data Specifies additional data to be passed to the specified procedure when it is called.
time Specifies the timestamp that indicates when the selection request was initiated. This should be the timestamp of the event that triggered this request; the value CurrentTime is not acceptable.

The XtGetSelectionValue function requests the value of the selection converted to the target type. The specified callback will be called at some time after XtGetSelectionValue is called, when the selection data is received from the X server. It may be called before or after XtGetSelectionValue returns. For more information about selection, target, and time, see Section 2.6 in the Inter-Client Communication Conventions Manual.

void XtGetSelectionValues(w, selection, targets, count, callback, client_data, time)
  Widget w;
  Atom selection;
  Atom *targets;
  int count;
  XtSelectionCallbackProc callback;
  XtPointer *client_data;
  Time time;

w Specifies the widget making the request. Must be of class Core or any subclass thereof.

selection Specifies the particular selection desired (that is, primary or secondary).
targets Specifies the types of information needed about the selection.
count Specifies the length of the targets and client_data lists.
callback Specifies the callback procedure to be called with each selection value obtained. Note that this is how the selection values are communicated back to the client.

client_data Specifies a list of additional data values, one for each target type, that are passed to the callback procedure when it is called for that target.
time Specifies the timestamp that indicates when the selection request was initiated. This should be the timestamp of the event that triggered this request; the value CurrentTime is not acceptable.

The XtGetSelectionValues function is similar to multiple calls to XtGetSelectionValue except that it guarantees that no other client can assert ownership between requests and therefore that all the conversions will refer to the same selection value. The callback is invoked
once for each target value with the corresponding client data. For more information about selection, target, and time see Section 2.6 in the Inter-Client Communication Conventions Manual.

11.5.2.3. Setting the Selection Owner

To set the selection owner and indicate that the selection value will be provided in one piece, use XtOwnSelection.

Boolean XtOwnSelection(w, selection, time, convert_proc, lose_selection, done_proc)

- **w**: Specifies the widget that wishes to become the owner. Must be of class Core or any subclass thereof.
- **selection**: Specifies the name of the selection (for example, XA_PRIMARY).
- **time**: Specifies the timestamp that indicates when the ownership request was initiated. This should be the timestamp of the event that triggered ownership; the value CurrentTime is not acceptable.
- **convert_proc**: Specifies the procedure to be called whenever a client requests the current value of the selection.
- **lose_selection**: Specifies the procedure to be called whenever the widget has lost selection ownership, or NULL if the owner is not interested in being called back.
- **done_proc**: Specifies the procedure called after the requestor has received the selection value, or NULL if the owner is not interested in being called back.

The XtOwnSelection function informs the Intrinsics selection mechanism that a widget wishes to own a selection. It returns True if the widget successfully becomes the owner and False otherwise. The widget may fail to become the owner if some other widget has asserted ownership at a time later than this widget. The widget can lose selection ownership either because some other client asserted later ownership of the selection or because the widget voluntarily gave up ownership of the selection. The lose_selection procedure is not called if the widget fails to obtain selection ownership in the first place.

If a done_proc is specified, the client owns the storage allocated for passing the value to the Intrinsics. If done_proc is NULL, the convert_proc must allocate storage using XtMalloc, XtRealloc, or XtCalloc, and the value specified will be freed by the Intrinsics when the transfer is complete.

Usually, a selection owner maintains ownership indefinitely until some other client requests ownership, at which time the Intrinsics selection mechanism informs the previous owner that it has lost ownership of the selection. However, in response to some user actions (for example, when a user deletes the information selected), the application may wish to explicitly inform the Intrinsics that it no longer is to be the selection owner by using XtDisownSelection.

void XtDisownSelection(w, selection, time)

- **w**: Specifies the widget that wishes to relinquish ownership.
selection  Specifies the atom naming the selection being given up.
time  Specifies the timestamp that indicates when the request to relinquish selection ownership was initiated.

The XtDisownSelection function informs the Intrinsics selection mechanism that the specified widget is to lose ownership of the selection. If the widget does not currently own the selection, either because it lost the selection or because it never had the selection to begin with, XtDisownSelection does nothing.

After a widget has called XtDisownSelection, its convert procedure is not called even if a request arrives later with a timestamp during the period that this widget owned the selection. However, its done procedure will be called if a conversion that started before the call to XtDisownSelection finishes after the call to XtDisownSelection.

11.5.3. Using Incremental Transfers

When using the incremental interface, an owner may have to process more than one selection request for the same selection, converted to the same target, at the same time. The incremental functions take a request_id argument, which is an identifier that is guaranteed to be unique among all incremental requests that are active concurrently.

For example, consider the following:

- Upon receiving a request for the selection value, the owner sends the first segment.
- While waiting to be called to provide the next segment value but before sending it, the owner receives another request from a different requestor for the same selection value.
- To distinguish between the requests, the owner uses the request_id value. This allows the owner to distinguish between the first requestor, which is asking for the second segment, and the second requestor, which is asking for the first segment.

11.5.3.1. Incremental Transfer Procedures

The following procedures are used by selection owners who wish to provide the selection data in multiple segments.

The procedure pointer specified by the incremental owner to supply the selection data to the Intrinsics is of type XtConvertSelectionIncrProc.

typedef XtPointer XtRequestId;

typedef Boolean (*XtConvertSelectionIncrProc)(Widget, Atom*, Atom*, Atom*, XtPointer*, unsigned long*, int*, unsigned long*, XtPointer, XtRequestId*);

Widget w;
Atom *selection;
Atom *target;
Atom *type_return;
XtPointer *value_return;
unsigned long *length_return;
int *format_return;
unsigned long *max_length;
XtPointer client_data;
XtRequestId *request_id;

w  Specifies the widget that currently owns this selection.
selection  Specifies the atom that names the selection requested.
target Specifies the type of information required about the selection.

type_return Specifies a pointer to an atom into which the property type of the converted value of the selection is to be stored.

value_return Specifies a pointer into which a pointer to the converted value of the selection is to be stored. The selection owner is responsible for allocating this storage.

length_return Specifies a pointer into which the number of elements in value_return, each of size indicated by format_return, is to be stored.

format_return Specifies a pointer into which the size in bits of the data elements of the selection value is to be stored so that the server may byte-swap the data if necessary.

max_length Specifies the maximum number of bytes which may be transferred at any one time.

client_data Specifies the value passed in by the widget when it took ownership of the selection.

request_id Specifies an opaque identification for a specific request.

This procedure is called repeatedly by the Intrinsics selection mechanism to get the next incremental chunk of data from a selection owner who has called XtOwnSelectionIncremental. It must return True if the procedure has succeeded in converting the selection data or False otherwise. On the first call with a particular request id, the owner must begin a new incremental transfer for the requested selection and target. On subsequent calls with the same request id, the owner may assume that the previously supplied value is no longer needed by the Intrinsics; that is, a fixed transfer area may be allocated and returned in value_return for each segment to be transferred. This procedure should store a non-NULL value in value_return and zero in length_return to indicate that the entire selection has been delivered. After returning this final segment, the request id may be reused by the Intrinsics to begin a new transfer.

To retrieve the SelectionRequest event that triggered the selection conversion procedure, use XtGetSelectionRequest, described in Section 11.5.2.1.

The procedure pointer specified by the incremental selection owner when it desires notification upon no longer having ownership is of type XtLoseSelectionIncrProc.

typedef void (*XtLoseSelectionIncrProc)(Widget, Atom*, XtPointer);

Widget w;
Atom *selection;
XtPointer client_data;

w Specifies the widget that has lost the selection ownership.

selection Specifies the atom that names the selection.

client_data Specifies the value passed in by the widget when it took ownership of the selection.

This procedure, which is optional, is called by the Intrinsics to inform the selection owner that it no longer owns the selection.

The procedure pointer specified by the incremental selection owner when it desires notification of receipt of the data or when it manages the storage containing the data is of type XtSelectionDoneIncrProc.
typedef void (*XtSelectionDoneIncrProc)(Widget, Atom*, Atom*, XtRequestId*, XtPointer);
    Widget w;
    Atom *selection;
    Atom *target;
    XtRequestId *request_id;
    XtPointer client_data;

    w Specifies the widget that owns the selection.
    selection Specifies the atom that names the selection being transferred.
    target Specifies the target type to which the conversion was done.
    request_id Specifies an opaque identification for a specific request.
    client_data Specified the value passed in by the widget when it took ownership of the selection.

This procedure, which is optional, is called by the Intrinsics after the requestor has retrieved the final (zero-length) segment of the incremental transfer to indicate that the entire transfer is complete. If this procedure is not specified, the Intrinsics will free only the final value returned by the selection owner using XtFree.

The procedure pointer specified by the incremental selection owner to notify it if a transfer should be terminated prematurely is of type XtCancelConvertSelectionProc.

typedef void (*XtCancelConvertSelectionProc)(Widget, Atom*, Atom*, XtRequestId*, XtPointer);
    Widget w;
    Atom *selection;
    Atom *target;
    XtRequestId *request_id;
    XtPointer client_data;

    w Specifies the widget that owns the selection.
    selection Specifies the atom that names the selection being transferred.
    target Specifies the target type to which the conversion was done.
    request_id Specifies an opaque identification for a specific request.
    client_data Specifies the value passed in by the widget when it took ownership of the selection.

This procedure is called by the Intrinsics when it has been determined by means of a timeout or other mechanism that any remaining segments of the selection no longer need to be transferred. Upon receiving this callback, the selection request is considered complete and the owner can free the memory and any other resources that have been allocated for the transfer.

11.5.3.2. Getting the Selection Value Incrementally
To obtain the value of the selection using incremental transfers, use XtGetSelectionValueIncremental or XtGetSelectionValuesIncremental.

void XtGetSelectionValueIncremental(w, selection, target, selection_callback, client_data, time)
    Widget w;
    Atom selection;
    Atom target;
    XtSelectionCallbackProc selection_callback;
    XtPointer client_data;
    Time time;

    w Specifies the widget making the request. Must be of class Core or any subclass thereof.
The XtGetSelectionValuesIncremental function is similar to XtGetSelectionValue except that the selection callback procedure will be called repeatedly upon delivery of multiple segments of the selection value. The end of the selection value is indicated when selection_callback is called with a non-NULL value of length zero, which must still be freed by the client. If the transfer of the selection is aborted in the middle of a transfer (for example, because to timeout), the selection_callback procedure is called with a type value equal to the symbolic constant XT_CONVERT_FAIL so that the requestor can dispose of the partial selection value it has collected up until that point. Upon receiving XT_CONVERT_FAIL, the requesting client must determine for itself whether or not a partially completed data transfer is meaningful. For more information about selection, target, and time, see Section 2.6 in the Inter-Client Communication Conventions Manual.

```c
void XtGetSelectionValuesIncremental(
    Widget w;
    Atom selection;
    Atom *targets;
    int count;
    XtSelectionCallbackProc selection_callback;
    XtPointer *client_data;
    Time time;
)
```

- **w** Specifies the widget making the request. Must be of class Core or any subclass thereof.
- **selection** Specifies the particular selection desired.
- **targets** Specifies the types of information needed about the selection.
- **count** Specifies the length of the targets and client_data lists.
- **selection_callback** Specifies the callback procedure to be called to receive each selection value.
- **client_data** Specifies a list of client data (one for each target type) values that are passed to the callback procedure when it is invoked for the corresponding target.
- **time** Specifies the timestamp that indicates when the selection request was initiated. This should be the timestamp of the event that triggered this request; the value CurrentTime is not acceptable.

The XtGetSelectionValuesIncremental function is similar to XtGetSelectionValueIncremental except that it takes a list of targets and client data. XtGetSelectionValuesIncremental is equivalent to calling XtGetSelectionValueIncremental successively for each target/client_data pair except that XtGetSelectionValuesIncremental does guarantee that all the conversions will use the same selection value because the ownership of the selection cannot change in the middle of the list, as would be possible when calling XtGetSelectionValueIncremental repeatedly. For more information about selection, target, and time, see Section 2.6 in the Inter-Client Communication Conventions Manual.
11.5.3.3. Setting the Selection Owner for Incremental Transfers

To set the selection owner when using incremental transfers, use `XtOwnSelectionIncremental`.

```c
Boolean XtOwnSelectionIncremental(w, selection, time, convert_callback, lose_callback, done_callback, cancel_callback, client_data)
```

- `w`: Specifies the widget that wishes to become the owner. Must be of class Core or any subclass thereof.
- `selection`: Specifies the atom that names the selection.
- `time`: Specifies the timestamp that indicates when the selection ownership request was initiated. This should be the timestamp of the event that triggered ownership; the value `CurrentTime` is not acceptable.
- `convert_callback`: Specifies the procedure to be called whenever the current value of the selection is requested.
- `lose_callback`: Specifies the procedure to be called whenever the widget has lost selection ownership, or NULL if the owner is not interested in being notified.
- `done_callback`: Specifies the procedure called after the requestor has received the entire selection, or NULL if the owner is not interested in being notified.
- `cancel_callback`: Specifies the callback procedure to be called when a selection request aborts because a timeout expires, or NULL if the owner is not interested in being notified.
- `client_data`: Specifies the argument to be passed to each of the callback procedures when they are called.

The `XtOwnSelectionIncremental` procedure informs the Intrinsics incremental selection mechanism that the specified widget wishes to own the selection. It returns `True` if the specified widget successfully becomes the selection owner or `False` otherwise. For more information about `selection`, `target`, and `time`, see Section 2.6 in the `Inter-Client Communication Conventions Manual`.

If a `done_callback` procedure is specified, the client owns the storage allocated for passing the value to the Intrinsics. If `done_callback` is NULL, the `convert_callback` procedure must allocate storage using `XtMalloc`, `XtRealloc`, or `XtCalloc`, and the final value specified will be freed by the Intrinsics when the transfer is complete. After a selection transfer has started, only one of the `done_callback` or `cancel_callback` procedures will be invoked to indicate completion of the transfer.

The `lose_callback` procedure does not indicate completion of any in-progress transfers; it will be invoked at the time a `SelectionClear` event is dispatched regardless of any active transfers, which are still expected to continue.

A widget that becomes the selection owner using `XtOwnSelectionIncremental` may use `XtDisownSelection` to relinquish selection ownership.
11.5.4. Retrieving the Most Recent Timestamp

To retrieve the timestamp from the most recent call to `XtDispatchEvent` that contained a timestamp, use `XtLastTimestampProcessed`.

```c
Time XtLastTimestampProcessed(display)
    Display *display;
```

`display` Specifies an open display connection.

If no `KeyPress`, `KeyRelease`, `ButtonPress`, `ButtonRelease`, `MotionNotify`, `EnterNotify`, `LeaveNotify`, `PropertyNotify`, or `SelectionClear` event has yet been passed to `XtDispatchEvent` for the specified display, `XtLastTimestampProcessed` returns zero.

11.6. Merging Exposure Events into a Region

The Intrinsics provide an `XtAddExposureToRegion` utility function that merges `Expose` and `GraphicsExpose` events into a region for clients to process at once rather than processing individual rectangles. For further information about regions, see Section 16.5 in `Xlib - C Language X Interface`.

To merge `Expose` and `GraphicsExpose` events into a region, use `XtAddExposureToRegion`.

```c
void XtAddExposureToRegion(event, region)
    XEvent *event;
    Region region;
```

`event` Specifies a pointer to the `Expose` or `GraphicsExpose` event.

`region` Specifies the region object (as defined in `<XI1/Xutil.h>`).

The `XtAddExposureToRegion` function computes the union of the rectangle defined by the exposure event and the specified region. Then it stores the results back in `region`. If the event argument is not an `Expose` or `GraphicsExpose` event, `XtAddExposureToRegion` returns without an error and without modifying `region`.

This function is used by the exposure compression mechanism; see Section 7.9.3.

11.7. Translating Widget Coordinates

To translate an x-y coordinate pair from widget coordinates to root window absolute coordinates, use `XtTranslateCoords`.

```c
void XtTranslateCoords(w, x, y, rootx_return, rooty_return)
    Widget w;
    Position x, y;
    Position *rootx_return, *rooty_return;
```

`w` Specifies the widget. Must be of class RectObj or any subclass thereof.

`x` `y` Specify the widget-relative x and y coordinates.

`rootx_return` `rooty_return` Return the root-relative x and y coordinates.

While `XtTranslateCoords` is similar to the Xlib `XTranslateCoordinates` function, it does not generate a server request because all the required information already is in the widget’s data structures.
11.8. Translating a Window to a Widget

To translate a given window and display pointer into a widget instance, use XtWindowToWidget.

Widget XtWindowToWidget(display, window)
  Display *display;
  Window window;

display  Specifies the display on which the window is defined.
window   Specifies the window for which you want the widget.

If there is a realized widget whose window is the specified window on the specified display, XtWindowToWidget returns that widget; otherwise, it returns NULL.

11.9. Handling Errors

The Intrinsics allow a client to register procedures that will be called whenever a fatal or non-fatal error occurs. These facilities are intended for both error reporting and logging and for error correction or recovery.

Two levels of interface are provided:

- A high-level interface that takes an error name and class and retrieves the error message text from an error resource database.
- A low-level interface that takes a simple string to display.

The high-level functions construct a string to pass to the lower-level interface. The strings may be specified in application code and will be overridden by the contents of an external system-wide file, the "error database file". The location and name of this file is implementation dependent.

Note

The application-context-specific error handling is not implemented on many systems, although the interfaces are always present. Most implementations will have just one set of error handlers for all application contexts within a process. If they are set for different application contexts, the ones registered last will prevail.

To obtain the error database (for example, to merge with an application- or widget-specific database), use XtAppGetErrorDatabase.

XrmDatabase *XtAppGetErrorDatabase(app_context)
  XtAppContext app_context;

app_context Specifies the application context.

The XtAppGetErrorDatabase function returns the address of the error database. The Intrinsics do a lazy binding of the error database and do not merge in the database file until the first call to XtAppGetErrorDatabaseText.

For a complete listing of all errors and warnings that can be generated by the Intrinsics, see Appendix D.

The high-level error and warning handler procedure pointers are of type XtErrorMsgHandler.
typedef void (*XtErrorMsgHandler)(String, String, String, String, String*, Cardinal*);

String name;     // Specifies the name to be concatenated with the specified type to form the resource name of the error message.
String type;     // Specifies the type to be concatenated with the name to form the resource name of the error message.
String class;    // Specifies the resource class of the error message.
String defaultp; // Specifies the default message to use if no error database entry is found.
String*params;   // Specifies a pointer to a list of parameters to be substituted in the message.
Cardinal*num_params;  // Specifies the number of entries in params.

The specified name can be a general kind of error, like "invalidParameters" or "invalidWindow", and the specified type gives extra information such as the name of the routine in which the error was detected. Standard printf notation is used to substitute the parameters into the message.

An error message handler can obtain the error database text for an error or a warning by calling XtAppGetErrorDatabaseText.

void XtAppGetErrorDatabaseText(app_context, name, type, class, default, buffer_return, nbytes, database);

XtAppContext app_context;  // Specifies the application context.
String name, type, class;  // Specify the name and type concatenated to form the resource name of the error message.
String default;            // Specifies the default message to use if no error database entry is found.
String buffer_return;      // Specifies the buffer into which the error message is to be returned.
int nbytes;                // Specifies the size of the buffer in bytes.
XrmDatabase database;      // Specifies the name of the alternative database to be used, or NULL if the application context's error database is to be used.

The XtAppGetErrorDatabaseText returns the appropriate message from the error database or returns the specified default message if one is not found in the error database. To form the full resource name and class when querying the database, the name and type are concatenated with a single "." between them and the class is concatenated with itself with a single "." if it does not already contain a ".".

To return the application name and class as passed to XtDisplayInitialize for a particular Display, use XtGetApplicationNameAndClass.
void XtGetApplicationNameAndClass(display, name_return, class_return)
Display* display;
String* name_return;
String* class_return;

display Specifies an open display connection that has been initialized with XtDisplay-
Initialize.

name_return Returns the application name.

class_return Returns the application class.

XtGetApplicationNameAndClass returns the application name and class passed to
XtDisplaylnitialize for the specified display. If the display was never initialized or has been
closed, the result is undefined. The returned strings are owned by the Intrinsics and must not
be modified or freed by the caller.

To register a procedure to be called on fatal error conditions, use XtAppSetEr-
rorMsgHandler.

XtErrorMsgHandler XtAppSetErrorMsgHandler(app_context, msg_handler)
XtAppContext app_context;
XtErrorMsgHandler msg_handler;

app_context Specifies the application context.

msg_handler Specifies the new fatal error procedure, which should not return.

XtAppSetErrorMsgHandler returns a pointer to the previously installed high-level fatal error
handler. The default high-level fatal error handler provided by the Intrinsics is named XtDe-
faultErrorMsg and constructs a string from the error resource database and calls XtError.
Fatal error message handlers should not return. If one does, subsequent Intrinsics behavior is
undefined.

To call the high-level error handler, use XtAppErrorMsg.

void XtAppErrorMsg(app_context, name, type, class, default, params, num_params)
XtAppContext app_context;
String name;
String type;
String class;
String default;
String *params;
Cardinal *num_params;

app_context Specifies the application context.

name Specifies the general kind of error.

type Specifies the detailed name of the error.

class Specifies the resource class.

default Specifies the default message to use if an error database entry is not found.

params Specifies a pointer to a list of values to be stored in the message.

num_params Specifies the number of entries in params.

The Intrinsics internal errors all have class "XtToolkitError".

To register a procedure to be called on nonfatal error conditions, use XtAppSetWar-
ningMsgHandler.
X Toolkit Intrinsics

XtErrorMsgHandler XtAppSetWarningMsgHandler(app_context, msg_handler)
    XtAppContext app_context;
    XtErrorMsgHandler msg_handler;

app_context Specifies the application context.
msg_handler Specifies the new nonfatal error procedure, which usually returns.

XtAppSetWarningMsgHandler returns a pointer to the previously installed high-level warning handler. The default high-level warning handler provided by the Intrinsics is named _XtDefaultWarningMsg and constructs a string from the error resource database and calls XtWarning.

To call the installed high-level warning handler, use XtAppWarningMsg.

void XtAppWarningMsg(app_context, name, type, class, default, params, num_params)
    XtAppContext app_context;
    String name;
    String type;
    String class;
    String default;
    String *params;
    Cardinal *num_params;

app_context Specifies the application context.
name Specifies the general kind of error.
type Specifies the detailed name of the error.
class Specifies the resource class.
default Specifies the default message to use if an error database entry is not found.
params Specifies a pointer to a list of values to be stored in the message.
num_params Specifies the number of entries in params.

The Intrinsics internal warnings all have class "XtToolkitError".

The low-level error and warning handler procedure pointers are of type XtErrorHandler.

typedef void (*XtErrorHandler)(String);
    String message;

message Specifies the error message.

The error handler should display the message string in some appropriate fashion.

To register a procedure to be called on fatal error conditions, use XtAppSetErrorHandler.

XtErrorHandler XtAppSetErrorHandler(app_context, handler)
    XtAppContext app_context;
    XtErrorHandler handler;

app_context Specifies the application context.
handler Specifies the new fatal error procedure, which should not return.

XtAppSetErrorHandler returns a pointer to the previously installed low-level fatal error handler. The default low-level error handler provided by the Intrinsics is _XtDefaultError. On POSIX-based systems, it prints the message to standard error and terminates the application. Fatal error message handlers should not return. If one does, subsequent Intrinsics behavior is undefined.
To call the installed fatal error procedure, use XtAppError:

```c
void XtAppError(app_context, message)
    XtAppContext app_context;
    String message;

app_context Specifies the application context.
message Specifies the message to be reported.
```

Most programs should use XtAppErrorMsg, not XtAppError, to provide for customization and internationalization of error messages.

To register a procedure to be called on nonfatal error conditions, use XtAppSetWarningHandler:

```c
XtErrorHandler XtAppSetWarningHandler(app_context, handler)
    XtAppContext app_context;
    XtErrorHandler handler;

app_context Specifies the application context.
handler Specifies the new nonfatal error procedure, which usually returns.
```

XtAppSetWarningHandler returns a pointer to the previously installed low-level warning handler. The default low-level warning handler provided by the Intrinsics is XtDefaultWarning. On POSIX-based systems, it prints the message to standard error and returns to the caller.

To call the installed nonfatal error procedure, use XtAppWarning:

```c
void XtAppWarning(app_context, message)
    XtAppContext app_context;
    String message;

app_context Specifies the application context.
message Specifies the nonfatal error message to be reported.
```

Most programs should use XtAppWarningMsg, not XtAppWarning, to provide for customization and internationalization of warning messages.

### 11.10. Setting WM_COLORMAP_WINDOWS

A client may set the value of the WM_COLORMAP_WINDOWS property on a widget’s window by calling XtSetWMColormapWindows:

```c
void XtSetWMColormapWindows(widget, list, count)
    Widget widget;
    Widget* list;
    Cardinal count;

widget Specifies the widget on whose window the WM_COLORMAP_WINDOWS property will be stored. Must be of class Core or any subclass thereof.
list Specifies a list of widgets whose windows are potentially to be listed in the WM_COLORMAP_WINDOWS property.
count Specifies the number of widgets in list.
```

XtSetWMColormapWindows returns immediately if `widget` is not realized or if `count` is 0. Otherwise, XtSetWMColormapWindows constructs an ordered list of windows by examining each widget in `list` in turn and ignoring the widget if it is not realized, or adding the widget’s window to the window list if the widget is realized and if its colormap resource is different.
from the colormap resources of all widgets whose windows are already on the window list.

Finally, \texttt{XtSetWMColormapWindows} stores the resulting window list in the \texttt{WM\_COLORMAP\_WINDOWS} property on the specified widget's window. Refer to Section 4.1.8 in the \textit{Inter-Client Communication Conventions Manual} for details of the semantics of the \texttt{WM\_COLORMAP\_WINDOWS} property.

\section*{11.11. Finding File Names}

The Intrinsics provide procedures to look for a file by name, allowing string substitutions in a list of file specifications. Two routines are provided for this: \texttt{XtFindFile} and \texttt{XtResolvePathname}. \texttt{XtFindFile} uses an arbitrary set of client-specified substitutions, and \texttt{XtResolvePathname} uses a set of standard substitutions corresponding to the \textit{X/Open Portability Guide} language localization conventions. Most applications should use \texttt{XtResolvePathname}.

A string substitution is defined by a list of \texttt{Substitution} entries.

\begin{verbatim}
typedef struct {
    char match;
    String substitution;
} SubstitutionRec, *Substitution;
\end{verbatim}

File name evaluation is handled in an operating-system-dependent fashion by an \texttt{XtFilePredicate} procedure.

\begin{verbatim}
typedef Boolean (*XtFilePredicate)(String);
\end{verbatim}

\begin{verbatim}
String filename;
\end{verbatim}

\texttt{filename} Specifies a potential filename.

A file predicate procedure will be called with a string that is potentially a file name. It should return \texttt{True} if this string specifies a file that is appropriate for the intended use and \texttt{False} otherwise.

To search for a file using substitutions in a path list, use \texttt{XtFindFile}.

\begin{verbatim}
String XtFindFile(par\_z, substitutions, num\_substitutions, predicate)
    String path;
    Substitution substitutions'
    Cardinal num\_substitutions';
    XtFilePredicate predicate'
\end{verbatim}

\begin{verbatim}
    path Specifies a path of file names, including substitution characters.
    substitutions Specifies a list of substitutions to make into the path.
    num\_substitutions Specifies the number of substitutions passed in.
    predicate Specifies a procedure called to judge each potential file name, or NULL.
\end{verbatim}

The \texttt{path} parameter specifies a string that consists of a series of potential file names delimited by colons. Within each name, the percent character specifies a string substitution selected by the following character. The character sequence "\%\%" specifies an embedded colon that is not a delimiter; the sequence is replaced by a single colon. The character sequence "\%" specifies a percent character that does not introduce a substitution; the sequence is replaced by a single percent character. If a percent character is followed by any other character, \texttt{XtFindFile} looks through the specified \texttt{substitutions} for that character in the \texttt{match} field and if found replaces the percent and match characters with the string in the corresponding \texttt{substitution} field. A \texttt{substitution} field entry of NULL is equivalent to a pointer to an empty string. If the operating system does not interpret multiple embedded name separators in the path (i.e., "\%/" in POSIX) the same way as a single separator, \texttt{XtFindFile} will collapse multiple separators into
a single one after performing all string substitutions. Except for collapsing embedded separators, the contents of the string substitutions are not interpreted by XtFindFile and may therefore contain any operating-system-dependent characters, including additional name separators. Each resulting string is passed to the predicate procedure until a string is found for which the procedure returns True; this string is the return value for XtFindFile. If no string yields a True return from the predicate, XtFindFile returns NULL.

If the predicate parameter is NULL, an internal procedure that checks if the file exists, is readable, and is not a directory will be used.

It is the responsibility of the caller to free the returned string using XtFree when it is no longer needed.

To search for a file using standard substitutions in a path list, use XtResolvePathname.

String XtResolvePathname(disp,ay, type, filename, suffix, path, substitutions, num_substitutions, predicate)

Display *display;
String type, filename, suffix, path;
Substitution substitutions;
Cardinal num_substitutions;
XtFilePredicate predicate;

display Specifies the display to use to find the language for language substitutions.
type filename suffix path
Specify values to substitute into the path.
path Specifies the list of file specifications, or NULL.
substitutions Specifies a list of additional substitutions to make into the path, or NULL.
num_substitutions Specifies the number of entries in substitutions.
predicate Specifies a procedure called to judge each potential file name, or NULL.

The substitutions specified by XtResolvePathname are determined from the value of the language string retrieved by XtDisplayInitialize for the specified display. To set the language for all applications specify "*xrtlLanguage: lang" in the resource database. The format and content of the language string are implementation-defined. One suggested syntax is to compose the language string of three parts: a "language part", a "territory part" and a "codeset part". The manner in which this composition is accomplished is implementation-defined and the Intrinsics make no interpretation of the parts other than to use them in substitutions as described below.

XtResolvePathname calls XtFindFile with the following substitutions in addition to any passed by the caller and returns the value returned by XtFindFile:

%N The value of the filename parameter, or the application’s class name if filename is NULL.
%T The value of the type parameter.
%S The value of the suffix parameter.
%L The language string associated with the specified display.
%l The language part of the display’s language string.
%t The territory part of the display’s language string.
%c The codeset part of the display’s language string.
%C The customization string retrieved from the resource database associated with display.

If a path is passed to XtResolvePathname, it will be passed along to XtFindFile. If the path argument is NULL, the value of the XFILESEARCHPATH environment variable will be passed to XtFindFile. If XFILESEARCHPATH is not defined, an implementation-specific
default path will be used which contains at least 6 entries. These entries must contain the following substitutions:

1. %C, %N, %S, %T, %L or %C, %N, %S, %T, %1, %t, %c
2. %C, %N, %S, %T, %1
3. %C, %N, %S, %T
4. %N, %S, %T, %L or %N, %S, %T, %1, %t, %c
5. %N, %S, %T, %1
6. %N, %S, %T

The order of these six entries within the path must be as given above. The order and use of substitutions within a given entry is implementation dependent. If the path begins with a colon, it will be preceded by %N%S. If the path includes two adjacent colons, %N%S will be inserted between them.

The type parameter is intended to be a category of files, usually being translated into a directory in the pathname. Possible values might include "app-defaults", "help", and "bitmap".

The suffix parameter is intended to be appended to the file name. Possible values might include ".txt", ".dat", and ".bm".

A suggested value for the default path on POSIX-based systems is

```
/usr/lib/X11/%L/%T/%N%C%S:/usr/lib/X11/%V/%T/%N%C%S:
/usr/lib/X11/%T/%N%C%S:/usr/lib/X11/%L/%T/%N%S:
/usr/lib/X11/%V/%T/%N%S:/usr/lib/X11/%T/%N%S
```

Using this example, if the user has specified a language, it will be used as a subdirectory of /usr/lib/X11 that will be searched for other files. If the desired file is not found there, the lookup will be tried again using just the language part of the specification. If the file is not there, it will be looked for in /usr/lib/X11. The type parameter is used as a subdirectory of the language directory or of /usr/lib/X11, and suffix is appended to the file name.

The customization string is obtained by querying the resource database currently associated with the display (the database returned by XrmGetDatabase) for the resource application_name.customization, class application_class.Customization where application_name and application_class are the values returned by XtGetApplicationNameAndClass. If no value is specified in the database, the empty string is used.

It is the responsibility of the caller to free the returned string using XtFree when it is no longer needed.
Chapter 12
Nonwidget Objects

Although widget writers are free to treat Core as the base class of the widget hierarchy, there are actually three classes above it. These classes are Object, RectObj, (Rectangle Object) and (unnamed) and members of these classes are referred to generically as objects. By convention, the term widget refers only to objects that are a subclass of Core, and the term nonwidget refers to objects that are not a subclass of Core. In the preceding portion of this specification, the interface descriptions indicate explicitly whether the generic widget argument is restricted to particular subclasses of Object. Sections 12.2.5, 12.3.5, and 12.5 summarize the permissible classes of the arguments to, and return values from, each of the Intrinsics routines.

12.1. Data Structures
In order not to conflict with previous widget code, the data structures used by nonwidget objects do not follow all the same conventions as those for widgets. In particular, the class records are not composed of parts but instead are complete data structures with filler for the widget fields they do not use. This allows the static class initializers for existing widgets to remain unchanged.

12.2. Object Objects
The Object object contains the definitions of fields common to all objects. It encapsulates the mechanisms for resource management. All objects and widgets are members of subclasses of Object, which is defined by the ObjectClassPart and ObjectPart structures.

12.2.1. ObjectClassPart Structure
The common fields for all object classes are defined in the ObjectClassPart structure. All fields have the same purpose, function, and restrictions as the corresponding fields in CoreClassPart; fields whose names are obj/n for some integer n are not used for Object, but exist to pad the data structure so that it matches Core's class record. The class record initialization must fill all obj/n fields with NULL or zero as appropriate to the type.

typedef struct _ObjectClassPart {
    WidgetClass superclass;
    String class_name;
    Cardinal widget_size;
    XtProc class_initialize;
    XtWidgetClassProc class_part_initialize;
    XtEnum class_inited;
    XtInitProc initialize;
    XtArgsProc initialize_hook;
    XtProc obj1;
    XtPointer obj2;
    Cardinal obj3;
    XtResourceList resources;
    Cardinal num_resources;
}
The prototypical ObjectClass consists of just the ObjectClassPart.

typedef struct _ObjectClassRec {
    ObjectClassPart object_class;
} ObjectClassRec, *ObjectClass;

The predefined class record and pointer for ObjectClassRec are
In IntrinsicP.h:
extern ObjectClassRec objectClassRec;

In Intrinsic.h:
extern WidgetClass objectClass;

The opaque types Object and ObjectClass and the opaque variable objectClass are defined for generic actions on objects. Intrinsic.h uses an incomplete structure definition to ensure that the compiler catches attempts to access private data:
typedef struct _ObjectClassRec* ObjectClass;

12.2.2. ObjectPart Structure

The common fields for all object instances are defined in the ObjectPart structure. All fields have the same meaning as the corresponding fields in CorePart.

typedef struct _ObjectPart {
    Widget self;
    WidgetClass widget_class;
    Widget parent;
    Boolean being_destroyed;
    XtCallbackList destroy_callbacks;
    XtPointer constraints;
} ObjectPart;
All object instances have the Object fields as their first component. The prototypical type Object is defined with only this set of fields. Various routines can cast object pointers, as needed, to specific object types.

In IntrinsicP.h:
typedef struct _ObjectRec {
    ObjectPart object;
} ObjectRec, *Object;

In Intrinsic.h:
typedef struct _ObjectRec *Object;

12.2.3. Object Resources
The resource names, classes, and representation types specified in the objectClassRec resource list are

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNdestroyCallback</td>
<td>XtCCallback</td>
<td>XtRCallback</td>
</tr>
</tbody>
</table>

12.2.4. ObjectPart Default Values
All fields in ObjectPart have the same default values as the corresponding fields in CorePart.

12.2.5. Object Arguments To Intrinsics Routines
The WidgetClass arguments to the following procedures may be objectClass or any subclass:

XtlnitializeWidgetClass, XtCreateWidget, XtVaCreateWidget
XtIsSubclass, XtCheckSubclass
XtGetResourceList, XtGetConstraintResourceList

The Widget arguments to the following procedures may be of class Object or any subclass:

XtCreateWidget, XtVaCreateWidget
XtAddCallback, XtAddCallbacks, XtRemoveCallback, XtRemoveCallbacks,
XtRemoveAllCallbacks, XtCallCallbacks, XtHasCallbacks, XtCallCallbackList
XtClass, XtSuperclass, XtIsSubclass, XtCheckSubclass, XtIsObject, XtIsRectObj,
XtIsWidget, XtIsComposite, XtIsConstraint, XtIsShell, XtIsOverrideShell,
XtIsWMShell, XtIsVendorShell, XtIsTransientShell, XtIsToplevelShell, XtIsApplicationShell.
XtIsManaged, XtIsSensitive
(both will return False if argument is not a subclass of RectObj)
XtIsRealized
(retuns the state of the nearest windowed ancestor if class of argument is not a subclass of Core)
The return value of the following procedures will be of class Object or a subclass:

XtCreateWidget, XtVaCreateWidget
XtParent
XtNameToWidget

The return value of the following procedures will be objectClass or a subclass:

XtClass, XtSuperclass

12.2.6. Use of Objects

The Object class exists to enable programmers to use the Intrinsics' classing and resource-handling mechanisms for things smaller and simpler than widgets. Objects make obsolete many common uses of subresources as described in sections 9.4, 9.7.2.4, and 9.7.2.5.

Composite widget classes that wish to accept nonwidget children must set the accepts_objects field in the CompositeClassExtension structure to True. XtCreateWidget will otherwise generate an error message on an attempt to create a nonwidget child.

Of the classes defined by the Intrinsics, only ApplicationShell accepts nonwidget children, and the class of any nonwidget child must not be rectObjClass or any subclass. The intent of allowing Object children of ApplicationShell is to provide clients a simple mechanism for establishing the resource-naming root of an object hierarchy.

12.3. Rectangle Objects

The class of rectangle objects is a subclass of Object that represents rectangular areas. It encapsulates the mechanisms for geometry management, and is called RectObj to avoid conflict with the Xlib Rectangle data type.

12.3.1. RectObjClassPart Structure

As with the ObjectClassPart structure, all fields in the RectObjClassPart structure have the same purpose and function as the corresponding fields in CoreClassPart; fields whose names are rectn for some integer n are not used for RectObj but exist to pad the data structure so that it matches Core's class record. The class record initialization must fill all rectn fields with NULL or zero as appropriate to the type.

typedef struct _RectObjClassPart {
    WidgetClass superclass;
    WidgetClass superclass;
    } RectObjClassPart;
String class_name;
Cardinal widget_size;
XtProc class_initialize;
XtWidgetClassProc class_part_initiaIize;
XtEnum class_initialized;
XtInitProc initialize;
XtArgsProc initialize_hook;
XtProc rect1;
XtPointer rect2;
Cardinal rect3;
XtResourceList resources;
Cardinal num_resources;
XrmClass xrm_class;
Boolean rect4;
XtEnum rect5;
Boolean rect6;
Boolean rect7;
XtWidgetProc destroy;
XtWidgetProc resize;
XtExposeProc expose;
XtSetValuesFunc set_values;
XtArgsFunc set_values_hook;
XtAlmostProc set_values_almost;
XtArgsProc get_values_hook;
XtProc rect9;
XtVersionType version;
XtPointer callback_private;
String rect10;
XtGeometryHandler query_geometry;
XtProc rect11;
XtPointer extension;
}

The RectObj class record consists of just the RectObjClassPart.

typedef struct _RectObjClassRec {
  RectObjClassPart rect_class;
} RectObjClassRec, *RectObjClass;

The predefined class record and pointer for RectObjClassRec are

In Intrinsic.h:
extern RectObjClassRec rectObjClassRec;

In Intrinsic.h:
extern WidgetClass rectObjClass;

The opaque types RectObj and RectObjClass and the opaque variable rectObjClass are
defined for generic actions on objects whose class is RectObj or a subclass of RectObj.
Intrinsic.h uses an incomplete structure definition to ensure that the compiler catches attempts
to access private data:

typedef struct _RectObjClassRec* RectObjClass;
12.3.2. RectObjPart Structure
In addition to the ObjectPart fields, RectObj objects have the following fields defined in the RectObjPart structure. All fields have the same meaning as the corresponding field in CorePart.

typedef struct _RectObjPart {
    Position x, y;
    Dimension width, height;
    Dimension border_width;
    Boolean managed;
    Boolean sensitive;
    Boolean ancestor_sensitive;
} RectObjPart;

RectObj objects have the RectObj fields immediately following the Object fields.

typedef struct _RectObjRec {
    ObjectPart object;
    RectObjPart rectangle;
} RectObjRec, *RectObj;

In Intrinsic.h:
typedef struct _RectObjRec* RectObj;

12.3.3. RectObj Resources
The resource names, classes, and representation types that are specified in the RectObjClassRec resource list are

<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNancestorSensitive</td>
<td>XtCSensitive</td>
<td>XtRBoolean</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>XtCBorderWidth</td>
<td>XtRDimension</td>
</tr>
<tr>
<td>XtNheight</td>
<td>XtCHeight</td>
<td>XtRDimension</td>
</tr>
<tr>
<td>XtNsensitive</td>
<td>XtCSensitive</td>
<td>XtRBoolean</td>
</tr>
<tr>
<td>XtNwidth</td>
<td>XtCWidth</td>
<td>XtRDimension</td>
</tr>
<tr>
<td>XtNx</td>
<td>XtCPosition</td>
<td>XtRPosition</td>
</tr>
<tr>
<td>XtNy</td>
<td>XtCPosition</td>
<td>XtRPosition</td>
</tr>
</tbody>
</table>

12.3.4. RectObjPart Default Values
All fields in RectObjPart have the same default values as the corresponding fields in CorePart.

12.3.5. Widget Arguments To Intrinsics Routines
The WidgetClass arguments to the following procedures may be rectObjClass or any subclass:

XtCreateManagedWidget, XtVaCreateManagedWidget

The Widget arguments to the following procedures may be of class RectObj or any subclass:
X Toolkit Intrinsics

12.3.6. Use of Rectangle Objects

RectObj can be subclassed to provide widgetlike objects (sometimes called gadgets) that do not use windows and do not have features often unused in simple widgets. This can save memory resources both in the server and in applications but requires additional support code in the parent. In the following discussion, rectobj refers only to objects whose class is RectObj or a subclass of RectObj but not Core or a subclass of Core.

Composite widget classes that wish to accept rectobj children must set the accepts_objects field in the CompositeClassExtension extension structure to True. XtCreateWidget or XtCreateManagedWidget will otherwise generate an error if called to create a nonwidget child. If the composite widget supports only children of class RectObj or a subclass (i.e., not of the general Object class), it must declare an insert_child procedure and check the subclass of each new child in that procedure. None of the classes defined by the Intrinsics accept rectobj children.

If gadgets are defined in an object set, the parent is responsible for much more than the parent of a widget. The parent must request and handle input events that occur for the gadget and is responsible for making sure that when it receives an exposure event the gadget children get drawn correctly. Rectobj children may have expose procedures specified in their class records, but the parent is free to ignore them, instead drawing the contents of the child itself. This can potentially save graphics context switching. The precise contents of the exposure event and region arguments to the RectObj expose procedure are not specified by the Intrinsics; a particular rectangle object is free to define the coordinate system origin (self-relative or parent-relative) and whether or not the rectangle or region is assumed to have been intersected with the visible region of the object.

In general, it is expected that a composite widget that accepts nonwidget children will document those children it is able to handle, since a gadget cannot be viewed as a completely self-contained entity, as can a widget. Since a particular composite widget class is usually designed to handle nonwidget children of only a limited set of classes, it should check the classes of newly added children in its insert_child procedure to make sure that it can deal with them.

The Intrinsics will clear areas of a parent window obscured by rectobj children, causing exposure events, under the following circumstances:

- A rectobj child is managed or unmanaged.
- In a call to XtSetValues on a rectobj child, one or more of the set_values procedures returns True.
- In a call to XtConfigureWidget on a rectobj child, areas will be cleared corresponding to both the old and the new child geometries, including the border, if the geometry changes.
• In a call to XtMoveWidget on a rectobj child, areas will be cleared corresponding to both the old and the new child geometries, including the border, if the geometry changes.

• In a call to XtResizeWidget on a rectobj child, an single rectangle will be cleared corresponding to the larger of the old and the new child geometries if they are different.

• In a call to XtMakeGeometryRequest (or XtMakeResizeRequest) on a rectobj child with XtQueryOnly not set, if the manager returns XtGeometryYes, two rectangles will be cleared corresponding to both the old and the new child geometries.

Stacking order is not supported for rectobj children. Composite widgets with rectobj children are free to define any semantics desired if the child geometries overlap, including making this an error.

When a rectobj is playing the role of a widget, developers must be reminded to avoid making assumptions about the object passed in the Widget argument to a callback procedure.

12.4. Undeclared Class
The Intrinsics define an unnamed class between RectObj and Core for possible future use by the X Consortium. The only assumptions that may be made about the unnamed class are

• the core_class.superclass field of coreWidgetClassRec contains a pointer to the unnamed class record.

• a pointer to the unnamed class record when dereferenced as an ObjectClass will contain a pointer to rectObjClassRec in its object_class.superclass field.

Except for the above, the contents of the class record for this class and the result of an attempt to subclass or to create a widget of this unnamed class are undefined.

12.5. Widget Arguments To Intrinsics Routines
The WidgetClass arguments to the following procedures must be of class Shell or a subclass:

XtCreatePopupShell, XtVaCreatePopupShell, XtAppCreateShell, XtVaAppCreateShell

The Widget arguments to the following procedures must be of class Core or any subclass:

XtCreatePopupShell, XtVaCreatePopupShell
XtAddEventHandler, XtAddRawEventHandler, XtRemoveEventHandler, XtRemoveRawEventHandler, XtInsertEventHandler, XtInsertRawEventHandler
XtAddGrab, XtRemoveGrab, XtGrabKey, XtGrabKeyboard, XtUngrabKey, XtUngrabKeyboard, XtGrabButton, XtGrabPointer, XtUngrabButton, XtUngrabPointer
XtBuildEventMask
XtCreateWindow, XtDisplay, XtScreen, XtWindow
XtNameToWidget
XtGetSelectionValue, XtGetSelectionValues, XtOwnSelection, XtDisownSelection, XtOwnSelectionIncremental, XtGetSelectionValueIncremental, XtGetSelectionValuesIncremental, XtGetSelectionRequest
XtInstallAccelerators, XtInstallAllAccelerators (both destination and source)
XtAugmentTranslations, XtOverrideTranslations, XtUninstallTranslations, XtCallActionProc

180
X Toolkit Intrinsics

XtMapWidget, XtUnmapWidget
XtRealizeWidget, XtUnrealizeWidget
XtSetMappedWhenManaged
XtCallAcceptFocus, XtSetKeyboardFocus (subtree)
XtResizeWindow
XtSetWMColormapWindows

The Widget arguments to the following procedures must be of class Composite or any subclass:

XtCreateManagedWidget, XtVaCreateManagedWidget

The Widget arguments to the following procedures must be of a subclass of Shell:

XtPopdown, XtCallbackPopdown, XtPopup, XtCallbackNone, XtCallbackNonexclusive, XtCallbackExclusive, XtPopupSpringLoaded

The return value of the following procedure will be of class Core or a subclass:

XtWindowToWidget

The return value of the following procedures will be of a subclass of Shell:

XtAppCreateShell, XtVaAppCreateShell, XtAppInitialize, XtVaAppInitialize, XtCreatePopupShell, XtVaCreatePopupShell
Chapter 13

Evolution of The Intrinsics

The interfaces described by this specification have undergone several sets of revisions in the course of adoption as an X Consortium standard specification. Having now been adopted by the Consortium as a standard part of the X Window System, it is expected that this and future revisions will retain backward compatibility in the sense that fully conforming implementations of these specifications may be produced that provide source compatibility with widgets and applications written to previous Consortium standard revisions.

The Intrinsics do not place any special requirement on widget programmers to retain source or binary compatibility for their widgets as they evolve, but several conventions have been established to assist those developers who want to provide such compatibility.

In particular, widget programmers may wish to conform to the convention described in Section 1.6.12 when defining class extension records.

13.1. Determining Specification Revision Level

Widget and application developers who wish to maintain a common source pool that will build properly with implementations of the Intrinsics at different revision levels of these specifications but that take advantage of newer features added in later revisions may use the symbolic macro XtSpecificationRelease.

#define XtSpecificationRelease 5

As the symbol XtSpecificationRelease was new to Release 4, widgets and applications desiring to build against earlier implementations should test for the presence of this symbol and assume only Release 3 interfaces if the definition is not present.

13.2. Release 3 to Release 4 Compatibility

At the data structure level, Release 4 retains binary compatibility with Release 3 (the first X Consortium standard release) for all data structures except WMShellPart, TopLevelShellPart, and TransientShellPart. Release 4 changed the argument type to most procedures that now take arguments of type XtPointer and structure members that are now of type XtPointer in order to avoid potential ANSI C conformance problems. It is expected that most implementations will be binary compatible with the previous definition.

Two fields in CoreClassPart were changed from Boolean to XtEnum to allow implementations additional freedom in specifying the representations of each. This change should require no source modification.

13.2.1. Additional Arguments

Arguments were added to the procedure definitions for XtInitProc, XtSetValuesFunc, and XtEventHandler to provide more information and to allow event handlers to abort further dispatching of the current event (caution is advised!). The added arguments to XtInitProc and XtSetValuesFunc make the initialize_hook and set_values_hook methods obsolete, but the hooks have been retained for those widgets that used them in Release 3.
13.2.2. `set_values_almost` Procedures

The use of the arguments by a `set_values_almost` procedure was poorly described in Release 3 and was inconsistent with other conventions.

The current specification for the manner in which a `set_values_almost` procedure returns information to the Intrinsics is not compatible with the Release 3 specification, and all widget implementations should verify that any `set_values_almost` procedures conform to the current interface.

No known implementation of the Intrinsics correctly implemented the Release 3 interface, so it is expected that the impact of this specification change is small.

13.2.3. Query Geometry

A composite widget layout routine that calls `XtQueryGeometry` is now expected to store the complete new geometry in the intended structure; previously the specification said "store the changes it intends to make". Only by storing the complete geometry does the child have any way to know what other parts of the geometry may still be flexible. Existing widgets should not be affected by this, except to take advantage of the new information.

13.2.4. `unrealizeCallback` Callback List

In order to provide a mechanism for widgets to be notified when they become unrealized through a call to `XtUnrealizeWidget`, the callback list name "unrealizeCallback" has been defined by the Intrinsics. A widget class that requires notification on unrealize may declare a callback list resource by this name. No class is required to declare this resource, but any class that did so in a prior revision may find it necessary to modify the resource name if it does not wish to use the new semantics.

13.2.5. Subclasses of WMShell

The formal adoption of the Inter-Client Communication Conventions Manual as an X Consortium standard has meant the addition of four fields to `WMShellPart` and one field to `TopLevelShellPart`. In deference to some widget libraries that had developed their own additional conventions to provide binary compatibility, these five new fields were added at the end of the respective data structures.

To provide more convenience for TransientShells, a field was added to the previously empty `TransientShellPart`. On some architectures the size of the part structure will not have changed as a result of this.

Any widget implementation whose class is a subclass of `TopLevelShell` or `TransientShell` must at minimum be recompiled with the new data structure declarations. Because `WMShellPart` no longer contains a contiguous `XSizeHints` data structure, a subclass that expected to do a single structure assignment of an `XSizeHints` structure to the `size_hints` field of `WMShellPart` must be revised, though the old fields remain at the same positions within `WMShellPart`.

13.2.6. Resource Type Converters

A new interface declaration for resource type converters was defined to provide more information to converters, to support conversion cache cleanup with resource reference counting, and to allow additional procedures to be declared to free resources. The old interfaces remain (in the compatibility section) and a new set of procedures was defined that work only with the new type converter interface.

In the now obsolete old type converter interface, converters are reminded that they must return the size of the converted value as well as its address. The example indicated this, but the
description of XtConverter was incomplete.

13.2.7. KeySym Case Conversion Procedure
The specification for the XtCaseProc function type has been changed to match the Release 3 implementation, which included necessary additional information required by the function (a pointer to the display connection), and corrects the argument type of the source KeySym parameter. No known implementation of the Intrinsics implemented the previously documented interface.

13.2.8. Nonwidget Objects
Formal support for nonwidget objects is new to Release 4. A prototype implementation was latent in at least one Release 3 implementation of the Intrinsics, but the specification has changed somewhat. The most significant change is the requirement for a composite widget to declare the CompositeClassExtension record with the accepts_objects field set to True in order to permit a client to create a nonwidget child.

The addition of this extension field ensures that composite widgets written under Release 3 will not encounter unexpected errors if an application attempts to create a nonwidget child. In Release 4 there is no requirement that all composite widgets implement the extra functionality required to manage windowless children, so the accept_objects field allows a composite widget to declare that it is not prepared to do so.

13.3. Release 4 to Release 5 Compatibility
At the data structure level, Release 5 retains complete binary compatibility with release 4. The specification of the ObjectPart, RectObjPart, CorePart, CompositePart, ShellPart, WMShellPart, TopLevelShellPart, and ApplicationShellPart instance records was made less strict to permit implementations to add internal fields to these structures. Any implementation that chooses to do so would, of course, force a recompilation. The Xlib specification for XrmValue and XrmOptionDescRec was updated to use a new type, XPointer, for the addr and value fields respectively, to avoid ANSI C conformance problems. The definition of XPointer is binary compatible with the previous implementation.

13.3.1. baseTranslations Resource
A new pseudo-resource, XtNbaseTranslations, was defined to permit application developers to specify translation tables in application defaults files while still giving end users the ability to augment or override individual event sequences. This change will affect only those applications that wish to take advantage of the new functionality, or those widgets that may have previously defined a resource named "baseTranslations".

Applications wishing to take advantage of the new functionality would change their application defaults file, e.g., from

```
app.widget.translations: value
```

to

```
app.widget.baseTranslations: value
```

If it is important to the application to preserve complete compatibility of the defaults file between different versions of the application running under Release 4 and Release 5, the full translations can be replicated in both the "translations" and the "baseTranslations" resource.
13.3.2. Resource File Search Path

The current specification allows implementations greater flexibility in defining the directory structure used to hold the application class and per-user application defaults files. Previous specifications required the substitution strings to appear in the default path in a certain order, preventing sites from collecting all the files for a specific application together in one directory. The Release 5 specification allows the default path to specify the substitution strings in any order within a single path entry. Users will need to pay close attention to the documentation for the specific implementation to know where to find these files and how to specify their own XFILESEARCHPATH and XUSERFILESEARCHPATH values when overriding the system defaults.

13.3.3. Customization Resource

XtResolvePathname supports a new substitution string, %C, for specifying separate application class resource files according to arbitrary user-specified categories. The primary motivation for this addition was separate monochrome and color application class defaults files. The substitution value is obtained by querying the current resource database for the application resource name "customization", class "Customization". Any application that previously used this resource name and class will need to be aware of the possibly conflicting semantics.

13.3.4. Per-Screen Resource Database

To allow a user to specify separate preferences for each screen of a display, a per-screen resource specification string has been added and multiple resource databases are created; one for each screen. This will affect any application that modified the (formerly unique) resource database associated with the display subsequent to the Intrinsics database initialization. Such applications will need to be aware of the particular screen on which each shell widget is to be created.

Although the wording of the specification changed substantially in the description of the process by which the resource database(s) is initialized, the net effect is the same as in prior releases with the exception of the added per-screen resource specification and the new customization substitution string in XtResolvePathname.

13.3.5. Internationalization of Applications

Internationalization as defined by ANSI is a technology that allows support of an application in a single locale. In adding support for internationalization to the Intrinsics the restrictions of this model have been followed. In particular, the new Intrinsics interfaces are designed to not preclude an application from using other alternatives. For this reason, no Intrinsics routine makes a call to establish the locale. However, a convenience routine to establish the locale at initialize time has been provided, in the form of a default procedure that must be explicitly installed if the application desires ANSI C locale behavior.

As many objects in X, particularly resource databases, now inherit the global locale when they are created, applications wishing to use the ANSI C locale model should use the new function XtSetLanguageProc to do so.

The internationalization additions also define event filters as a part of the Xlib Input Method specifications. The Intrinsics enable the use of event filters through additions to XtDispatchEvent. Applications that may not be dispatching all events through
XtDispatchEvent should be reviewed in the context of this new input method mechanism. In order to permit internationalization of error messages the name and path of the error database file is now allowed to be implementation dependent. No adequate standard mechanism has yet been suggested to allow the Intrinsics to locate the database from localization information supplied by the client.

The previous specification for the syntax of the language string specified by xnilLanguage has been dropped to avoid potential conflicts with other standards. The language string syntax is now implementation-defined. The example syntax cited is consistent with the previous specification.

13.3.6. Permanently Allocated Strings

In order to permit additional memory savings, an Xlib interface was added to allow the resource manager to avoid copying certain string constants. The Intrinsics specification was updated to explicitly require the Object class name, resource name, resource class, resource type, default type in resource tables, Core actions string field, and Constraint resource name, resource class, resource type, and default type resource fields to be permanently allocated. This explicit requirement is expected to affect only applications that may create and destroy classes on the fly.

13.3.7. Arguments to Existing Functions

The args argument to XtAppInitialize, XtVaAppInitialize, XtOpenDisplay, XtDisplayInitialize, and XtInitialize were changed from Cardinal* to int* to conform to pre-existing convention and avoid otherwise annoying typecasting in ANSI C environments.
Appendix A

Resource File Format

A resource file contains text representing the default resource values for an application or set of applications.

The format of resource files is defined by *Xlib - C Language X Interface* and is reproduced here for convenience only.

The format of a resource specification is

```
ResourceLine    = Comment | IncludeFile | ResourceSpec | <empty line>
Comment         = '!' | {<any character except null or newline>}
IncludeFile     = '#' WhiteSpace 'include' WhiteSpace FileName WhiteSpace
FileName        = <valid filename for operating system>
ResourceSpec    = WhiteSpace ResourceName WhiteSpace ':' WhiteSpace Value
ResourceName    = [Binding] {Component Binding} ComponentName
Binding         = '..' | '..'
WhiteSpace      = {<space> | <horizontal tab>}
Component       = '..?' | ComponentName
ComponentName   = NameChar {NameChar}
NameChar        = 'a'..'z' | 'A'..'Z' | '0'..'9' | _ | ...
Value           = {<any character except null or unescaped newline>}
```

Elements separated by vertical bar (|) are alternatives. Curly braces ({...}) indicate zero or more repetitions of the enclosed elements. Square brackets ([...]) indicate that the enclosed element is optional. Quotes ("...") are used around literal characters.

If the last character on a line is a backslash (\), that line is assumed to continue on the next line.

To allow a Value to begin with whitespace, the two-character sequence "\space" (backslash followed by space) is recognized and replaced by a space character, and the two-character sequence "\tab" (backslash followed by horizontal tab) is recognized and replaced by a horizontal tab character.

To allow a Value to contain embedded newline characters, the two-character sequence "\n" is recognized and replaced by a newline character. To allow a Value to be broken across multiple lines in a text file, the two-character sequence "\newline" (backslash followed by newline) is recognized and removed from the value.

To allow a Value to contain arbitrary character codes, the four-character sequence "\nnnn", where each *n* is a digit character in the range of "0"-"7", is recognized and replaced with a single byte that contains the octal value specified by the sequence. Finally, the two-character sequence "\" is recognized and replaced with a single backslash.
Appendix B

Translation Table Syntax

Notation

Syntax is specified in EBNF notation with the following conventions:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ a ]</td>
<td>Means either nothing or &quot;a&quot;</td>
</tr>
<tr>
<td>{ a }</td>
<td>Means zero or more occurrences of &quot;a&quot;</td>
</tr>
<tr>
<td>( a</td>
<td>b )</td>
</tr>
<tr>
<td>\n</td>
<td>Is the newline character</td>
</tr>
</tbody>
</table>

All terminals are enclosed in double quotation marks (" "). Informal descriptions are enclosed in angle brackets (< >).

Syntax

The syntax of a translation table is:

```
translationTable = [ directive ] { production }
directive = ( "#replace" | "#override" | "#augment" ) "\n"
production = lhs "\n" rhs "\n"
 lhs = ( event | keyseq ) { ( event | keyseq ) }
keychar = """" keychar [ keychar ] ""
keyseq = [ "\" | "$" | "\" ] <ISO Latin 1 character>
event = [ modifier_list ] "<event_type>" [ "(" count ["+"] ")" ] ( detail )
modifier_list = ( ["1"] ["2"] ["3"] ["4"] ... ) modifier_name
count = ( ["1"] | ["2"] | ["3"] | ["4"] | ... )
modifier_name = "@" < keysym > | < see ModifierNames table below >
event_type = < see Event Types table below >
detail = < event specific details >
rhs = [ name "(" [ params ] ")" ]
name = namechar [ namechar ]
namechar = [ "a" | "z" | "A" | "z" | "0" | ... | "9" | _ | _ ]
params = string [ "\", string ]
string = quoted_string | unquoted_string
quoted_string = "\" [ < Latin 1 character > | escape_char | "\\" ] ""
escape_char = "\" 
unquoted_string = [ < Latin 1 character except space, tab, ",", "\", "\n", "" ]
```

The params field is parsed into a list of String values that will be passed to the named action procedure. A quoted string may contain an embedded quotation mark if the quotation mark is preceded by a single backslash (\). The three-character sequence "\\" is interpreted as "single backslash followed by end-of-string".

Modifier Names

The modifier field is used to specify standard X keyboard and button modifier mask bits. Modifiers are legal on event types KeyPress, KeyRelease, ButtonPress, ButtonRelease, MotionNotify, EnterNotify, LeaveNotify, and their abbreviations. An error is generated
when a translation table that contains modifiers for any other events is parsed.

- If the modifier list has no entries and is not "None", it means "don't care" on all modifiers.
- If an exclamation point (!) is specified at the beginning of the modifier list, it means that the listed modifiers must be in the correct state and no other modifiers can be asserted.
- If any modifiers are specified and an exclamation point (!) is not specified, it means that the listed modifiers must be in the correct state and "don't care" about any other modifiers.
- If a modifier is preceded by a tilde (~), it means that that modifier must not be asserted.
- If "None" is specified, it means no modifiers can be asserted.
- If a colon (:) is specified at the beginning of the modifier list, it directs the Intrinsics to apply any standard modifiers in the event to map the event keycode into a KeySym. The default standard modifiers are Shift and Lock, with the interpretation as defined in X Window System Protocol, Section 5. The resulting KeySym must exactly match the specified KeySym, and the nonstandard modifiers in the event must match the modifier list. For example, "!:<Key>a" is distinct from "!:<Key>A", and "!:Shift<Key>A" is distinct from "!:<Key>A"
- If both an exclamation point (!) and a colon (:) are specified at the beginning of the modifier list, it means that the listed modifiers must be in the correct state and that no other modifiers except the standard modifiers can be asserted. Any standard modifiers in the event are applied as for colon (:) above.
- If a colon (:) is not specified, no standard modifiers are applied. Then, for example, "<Key>A" and "<Key>a" are equivalent.

In key sequences, a circumflex (') is an abbreviation for the Control modifier, a dollar sign ($) is an abbreviation for Meta, and a backslash (\) can be used to quote any character, in particular a double quote ("), a circumflex ('), a dollar sign ($), and another backslash (\). Briefly:

| No Modifiers:                                      | None <event> detail |
| Any Modifiers:                                    | <event> detail      |
| Only these Modifiers:                             | ! mod1 mod2 <event> detail |
| These modifiers and any others:                   | mod1 mod2 <event> detail |

The use of "None" for a modifier list is identical to the use of an exclamation point with no modifiers.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl</td>
<td>c</td>
<td>Control modifier bit</td>
</tr>
<tr>
<td>Shift</td>
<td>s</td>
<td>Shift modifier bit</td>
</tr>
<tr>
<td>Lock</td>
<td>l</td>
<td>Lock modifier bit</td>
</tr>
<tr>
<td>Meta</td>
<td>m</td>
<td>Meta key modifier</td>
</tr>
<tr>
<td>Hyper</td>
<td>h</td>
<td>Hyper key modifier</td>
</tr>
<tr>
<td>Super</td>
<td>su</td>
<td>Super key modifier</td>
</tr>
<tr>
<td>Alt</td>
<td>a</td>
<td>Alt key modifier</td>
</tr>
<tr>
<td>Mod1</td>
<td>Mod1 modifier bit</td>
<td></td>
</tr>
<tr>
<td>Mod2</td>
<td>Mod2 modifier bit</td>
<td></td>
</tr>
<tr>
<td>Mod3</td>
<td>Mod3 modifier bit</td>
<td></td>
</tr>
<tr>
<td>Mod4</td>
<td>Mod4 modifier bit</td>
<td></td>
</tr>
<tr>
<td>Mod5</td>
<td>Mod5 modifier bit</td>
<td></td>
</tr>
<tr>
<td>Button1</td>
<td>Button1 modifier bit</td>
<td></td>
</tr>
<tr>
<td>Button2</td>
<td>Button2 modifier bit</td>
<td></td>
</tr>
<tr>
<td>Button3</td>
<td>Button3 modifier bit</td>
<td></td>
</tr>
<tr>
<td>Button4</td>
<td>Button4 modifier bit</td>
<td></td>
</tr>
</tbody>
</table>
A key modifier is any modifier bit one of whose corresponding KeyCodes contains the corresponding left or right KeySym. For example, "m" or "Meta" means any modifier bit mapping to a KeyCode whose KeySym list contains XK_Meta_L or XK_Meta_R. Note that this interpretation is for each display, not global or even for each application context. The Control, Shift, and Lock modifier names refer explicitly to the corresponding modifier bits; there is no additional interpretation of KeySyms for these modifiers.

Because it is possible to associate arbitrary KeySyms with modifiers, the set of key modifiers is extensible. The "@" <keysym> syntax means any modifier bit whose corresponding KeyCode contains the specified KeySym name.

A modifier_list/KeySym combination in a translation matches a modifiers/KeyCode combination in an event in the following ways:

1. If a colon (:) is used, the Intrinsics call the display's XtKeyProc with the KeyCode and modifiers. To match, (modifiers & modifiers_return) must equal modifier_list, and keysym_return must equal the given KeySym.

2. If (:) is not used, the Intrinsics mask off all don't-care bits from the modifiers. This value must be equal to modifier_list. Then, for each possible combination of don't-care modifiers in the modifier list, the Intrinsics call the display's XtKeyProc with the KeyCode and that combination ORed with the cared-about modifier bits from the event. Keysym_return must match the KeySym in the translation.

Event Types

The event-type field describes XEvent types. In addition to the standard Xlib symbolic event type names, the following event type synonyms are defined:

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>KeyPress</td>
</tr>
<tr>
<td>KeyDown</td>
<td>KeyPress</td>
</tr>
<tr>
<td>KeyUp</td>
<td>KeyRelease</td>
</tr>
<tr>
<td>BtnDown</td>
<td>ButtonPress</td>
</tr>
<tr>
<td>BtnUp</td>
<td>ButtonRelease</td>
</tr>
<tr>
<td>Motion</td>
<td>MotionNotify</td>
</tr>
<tr>
<td>PtrMoved</td>
<td>MotionNotify</td>
</tr>
<tr>
<td>MouseMoved</td>
<td>MotionNotify</td>
</tr>
<tr>
<td>Enter</td>
<td>EnterNotify</td>
</tr>
<tr>
<td>EnterWindow</td>
<td>EnterNotify</td>
</tr>
<tr>
<td>Leave</td>
<td>LeaveNotify</td>
</tr>
<tr>
<td>LeaveWindow</td>
<td>LeaveNotify</td>
</tr>
<tr>
<td>FocusIn</td>
<td>FocusIn</td>
</tr>
<tr>
<td>FocusOut</td>
<td>FocusOut</td>
</tr>
<tr>
<td>Keymap</td>
<td>KeymapNotify</td>
</tr>
<tr>
<td>Expose</td>
<td>Expose</td>
</tr>
<tr>
<td>GrExp</td>
<td>GraphicsExpose</td>
</tr>
<tr>
<td>NoExp</td>
<td>NoExpose</td>
</tr>
<tr>
<td>Visible</td>
<td>VisibilityNotify</td>
</tr>
<tr>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Create</td>
<td>CreateNotify</td>
</tr>
<tr>
<td>Destroy</td>
<td>DestroyNotify</td>
</tr>
<tr>
<td>Unmap</td>
<td>UnmapNotify</td>
</tr>
<tr>
<td>Map</td>
<td>MapNotify</td>
</tr>
<tr>
<td>MapReq</td>
<td>MapRequest</td>
</tr>
<tr>
<td>Reparent</td>
<td>ReparentNotify</td>
</tr>
<tr>
<td>Configure</td>
<td>ConfigureNotify</td>
</tr>
<tr>
<td>ConfigureReq</td>
<td>ConfigureRequest</td>
</tr>
<tr>
<td>Grav</td>
<td>GravityNotify</td>
</tr>
<tr>
<td>ResReq</td>
<td>ResizeRequest</td>
</tr>
<tr>
<td>Circ</td>
<td>CirculateNotify</td>
</tr>
<tr>
<td>CircReq</td>
<td>CirculateRequest</td>
</tr>
<tr>
<td>Prop</td>
<td>PropertyNotify</td>
</tr>
<tr>
<td>SelClr</td>
<td>SelectionClear</td>
</tr>
<tr>
<td>SelReq</td>
<td>SelectionRequest</td>
</tr>
<tr>
<td>Select</td>
<td>SelectionNotify</td>
</tr>
<tr>
<td>Clrmap</td>
<td>ColormapNotify</td>
</tr>
<tr>
<td>Message</td>
<td>ClientMessage</td>
</tr>
<tr>
<td>Mapping</td>
<td>MappingNotify</td>
</tr>
</tbody>
</table>

The supported abbreviations are:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Event Type</th>
<th>Including</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl</td>
<td>KeyPress</td>
<td>with Control modifier</td>
</tr>
<tr>
<td>Meta</td>
<td>KeyPress</td>
<td>with Meta modifier</td>
</tr>
<tr>
<td>Shift</td>
<td>KeyPress</td>
<td>with Shift modifier</td>
</tr>
<tr>
<td>Btn1Down</td>
<td>ButtonPress</td>
<td>with Button1 detail</td>
</tr>
<tr>
<td>Btn1Up</td>
<td>ButtonRelease</td>
<td>with Button1 detail</td>
</tr>
<tr>
<td>Btn2Down</td>
<td>ButtonPress</td>
<td>with Button2 detail</td>
</tr>
<tr>
<td>Btn2Up</td>
<td>ButtonRelease</td>
<td>with Button2 detail</td>
</tr>
<tr>
<td>Btn3Down</td>
<td>ButtonPress</td>
<td>with Button3 detail</td>
</tr>
<tr>
<td>Btn3Up</td>
<td>ButtonRelease</td>
<td>with Button3 detail</td>
</tr>
<tr>
<td>Btn4Down</td>
<td>ButtonPress</td>
<td>with Button4 detail</td>
</tr>
<tr>
<td>Btn4Up</td>
<td>ButtonRelease</td>
<td>with Button4 detail</td>
</tr>
<tr>
<td>Btn5Down</td>
<td>ButtonPress</td>
<td>with Button5 detail</td>
</tr>
<tr>
<td>Btn5Up</td>
<td>ButtonRelease</td>
<td>with Button5 detail</td>
</tr>
<tr>
<td>BtnMotion</td>
<td>MotionNotify</td>
<td>with any button modifier</td>
</tr>
<tr>
<td>Btn1Motion</td>
<td>MotionNotify</td>
<td>with Button1 modifier</td>
</tr>
<tr>
<td>Btn2Motion</td>
<td>MotionNotify</td>
<td>with Button2 modifier</td>
</tr>
<tr>
<td>Btn3Motion</td>
<td>MotionNotify</td>
<td>with Button3 modifier</td>
</tr>
<tr>
<td>Btn4Motion</td>
<td>MotionNotify</td>
<td>with Button4 modifier</td>
</tr>
<tr>
<td>Btn5Motion</td>
<td>MotionNotify</td>
<td>with Button5 modifier</td>
</tr>
</tbody>
</table>

The detail field is event-specific and normally corresponds to the detail field of the corresponding event as described by *X Window System Protocol*, Section 11. The detail field is supported for the following event types:

| Event | Event Field       |
### X Toolkit Intrinsics

#### XI1 Release 5

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyPress</td>
<td>KeySym from event detail (keycode)</td>
</tr>
<tr>
<td>KeyRelease</td>
<td>KeySym from event detail (keycode)</td>
</tr>
<tr>
<td>ButtonPress</td>
<td>button from event detail</td>
</tr>
<tr>
<td>ButtonRelease</td>
<td>button from event detail</td>
</tr>
<tr>
<td>MotionNotify</td>
<td>event detail</td>
</tr>
<tr>
<td>EnterNotify</td>
<td>event mode</td>
</tr>
<tr>
<td>LeaveNotify</td>
<td>event mode</td>
</tr>
<tr>
<td>FocusIn</td>
<td>event mode</td>
</tr>
<tr>
<td>FocusOut</td>
<td>event mode</td>
</tr>
<tr>
<td>PropertyNotify</td>
<td>atom</td>
</tr>
<tr>
<td>SelectionClear</td>
<td>selection</td>
</tr>
<tr>
<td>SelectionRequest</td>
<td>selection</td>
</tr>
<tr>
<td>SelectionNotify</td>
<td>selection</td>
</tr>
<tr>
<td>ClientMessage</td>
<td>type</td>
</tr>
<tr>
<td>MappingNotify</td>
<td>request</td>
</tr>
</tbody>
</table>

If the event type is `KeyPress` or `KeyRelease`, the detail field specifies a KeySym name in standard format which is matched against the event as described above, for example, `<Key>A`.

For the `PropertyNotify`, `SelectionClear`, `SelectionRequest`, `SelectionNotify` and `ClientMessage` events the detail field is specified as an atom name; for example, `<Message>WM_PROTocols`. For the `MotionNotify`, `EnterNotify`, `LeaveNotify`, `FocusIn`, `FocusOut` and `MappingNotify` events, either the symbolic constants as defined by `X Window System Protocol`, Section 11, or the numeric values may be specified.

If no detail field is specified, then any value in the event detail is accepted as a match.

A KeySym can be specified as any of the standard KeySym names, a hexadecimal number prefixed with "0x" or "OX", an octal number prefixed with "0" or a decimal number. A KeySym expressed as a single digit is interpreted as the corresponding Latin 1 KeySym, for example, "0" is the KeySym XK_0. Other single character KeySyms are treated as literal constants from Latin 1, for example, "!" is treated as 0x21. Standard KeySym names are as defined in `<X11/keysymdef.h>` with the "XK_" prefix removed.

### Canonical Representation

Every translation table has a unique, canonical text representation. This representation is passed to a widget’s `display_accelerator` procedure to describe the accelerators installed on that widget. The canonical representation of a translation table is (see also “Syntax”)

```plaintext
translationTable = { production }
production = lhs "::" rhs "\n"
lhs = event { "," event }

event = [modifier_list] "<"event_type">" [ "," count["+""] "" ] { detail }
modifier_list = ["!" ] [ ":" ] (modifier)
modifier = ["*" ] modifier_name
count = ("1" | "2" | "3" | "4" | ...)
modifier_name = "@" <keysym> | <see canonical modifier names below>

event_type = <see canonical event types below>
detail = <event specific details>

rhs = { name "(" [params] ")" } 
name = namechar { namechar }
namechar = ("a" | "A" | "0" | ") " | ") " | ") " ... |
params = string { "", " string }
string = quoted_string
quoted_string = "" [ <Latin 1 character> | escape_char ] "" |
escape_char = ""\""
```

---

192
The canonical modifier names are

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>Button</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl</td>
<td>Mod1</td>
<td>Button1</td>
</tr>
<tr>
<td>Shift</td>
<td>Mod2</td>
<td>Button2</td>
</tr>
<tr>
<td>Lock</td>
<td>Mod3</td>
<td>Button3</td>
</tr>
<tr>
<td></td>
<td>Mod4</td>
<td>Button4</td>
</tr>
<tr>
<td></td>
<td>Mod5</td>
<td>Button5</td>
</tr>
</tbody>
</table>

The canonical event types are

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Event Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyPress</td>
<td>KeyRelease</td>
</tr>
<tr>
<td>ButtonPress</td>
<td>ButtonRelease</td>
</tr>
<tr>
<td>MotionNotify</td>
<td>EnterNotify</td>
</tr>
<tr>
<td>LeaveNotify</td>
<td>FocusIn</td>
</tr>
<tr>
<td>FocusOut</td>
<td>KeymapNotify</td>
</tr>
<tr>
<td>Expose</td>
<td>GraphicsExpose,</td>
</tr>
<tr>
<td>NoExpose</td>
<td>VisibilityNotify</td>
</tr>
<tr>
<td>CreateNotify</td>
<td>DestroyNotify</td>
</tr>
<tr>
<td>UnmapNotify</td>
<td>MapNotify</td>
</tr>
<tr>
<td>MapRequest</td>
<td>ReparentNotify</td>
</tr>
<tr>
<td>ConfigureNotify</td>
<td>ConfigureRequest</td>
</tr>
<tr>
<td>GravityNotify</td>
<td>ResizeRequest</td>
</tr>
<tr>
<td>CirculateNotify</td>
<td>CirculateRequest</td>
</tr>
<tr>
<td>PropertyNotify</td>
<td>SelectionClear</td>
</tr>
<tr>
<td>SelectionRequest</td>
<td>SelectionNotify</td>
</tr>
<tr>
<td>ColormapNotify</td>
<td>ClientMessage</td>
</tr>
</tbody>
</table>

Examples

- Always put more specific events in the table before more general ones:

  Shift <Btn1Down> : twas()
  <Btn1Down> : brillig()

- For double-click on Button1 Up with Shift, use this specification:

  Shift<Btn1Up>(2) : and()

  This is equivalent to the following line with appropriate timers set between events:

  Shift<Btn1Down>,Shift<Btn1Up>,Shift<Btn1Down>,Shift<Btn1Up> : and()

- For double-click on Button1 Down with Shift, use this specification:

  Shift<Btn1Down>(2) : the()

  This is equivalent to the following line with appropriate timers set between events:

  Shift<Btn1Down>,Shift<Btn1Up>,Shift<Btn1Down> : the()

- Mouse motion is always discarded when it occurs between events in a table where no motion event is specified:

  <Btn1Down>,<Btn1Up> : slithy()
This is taken, even if the pointer moves a bit between the down and up events. Similarly, any motion event specified in a translation matches any number of motion events. If the motion event causes an action procedure to be invoked, the procedure is invoked after each motion event.

- If an event sequence consists of a sequence of events that is also a noninitial subsequence of another translation, it is not taken if it occurs in the context of the longer sequence. This occurs mostly in sequences like the following:

  `<Btn1Down>,<Btn1Up>` : toves()\n  `<Btn1Up>` : did()

The second translation is taken only if the button release is not preceded by a button press or if there are intervening events between the press and the release. Be particularly aware of this when using the repeat notation, above, with buttons and keys, because their expansion includes additional events; and when specifying motion events, because they are implicitly included between any two other events. In particular, pointer motion and double-click translations cannot coexist in the same translation table.

- For single click on Button1 Up with Shift and Meta, use this specification:

  Shift Meta `<Btn1Down>` , Shift Meta `<Btn1Up>` : gyre()

- For multiple clicks greater or equal to a minimum number, a plus sign (+) may be appended to the final (rightmost) count in an event sequence. The actions will be invoked on the count-th click and each subsequent one arriving within the multi-click time interval. For example:

  Shift `<Btn1Up>(2+)` : and()

- To indicate `EnterNotify` with any modifiers, use this specification:

  `<Enter>` : gimble()

- To indicate `EnterNotify` with no modifiers, use this specification:

  None `<Enter>` : in()

- To indicate `EnterNotify` with Button1 Down and Button2 Up and "don't care" about the other modifiers, use this specification:

  `Button1 ~ Button2 <Enter>` : thc()

- To indicate `EnterNotify` with Button1 down and Button2 down exclusively, use this specification:

  `! Button1 Button2 <Enter>` : wabe()

You do not need to use a tilde (') with an exclamation point (!).
Appendix C

Compatibility Functions

In prototype versions of the X Toolkit each widget class implemented an Xt<Widget>Create (for example, XtLabelCreate) function, in which most of the code was identical from widget to widget. In the Intrinsics, a single generic XtCreateWidget performs most of the common work and then calls the initialize procedure implemented for the particular widget class.

Each Composite class also implemented the procedures Xt<Widget>Add and an Xt<Widget>Delete (for example, XtButtonBoxAddButton and XtButtonBoxDeleteButton). In the Intrinsics, the Composite generic procedures XtManageChildren and XtUnmanageChildren perform error checking and screening out of certain children. Then they call the change Managed procedure implemented for the widget’s Composite class. If the widget’s parent has not yet been realized, the call to the change Managed procedure is delayed until realization time.

Old style calls can be implemented in the X Toolkit by defining one-line procedures or macros that invoke a generic routine. For example, you could define the macro XtLabelCreate as:

```c
#define XtLabelCreate(name, parent, args, num_args) 
    ((LabelWidget) XtCreateWidget((name, labelWidgetClass, parent, args, num_args)))
```

Pop-up shells in some of the prototypes automatically performed an XtManageChild on their child within their insert_child procedure. Creators of pop-up children need to call XtManageChild themselves.

As a convenience to people converting from earlier versions of the toolkit without application contexts, the following routines exist: XtInitialize, XtMainLoop, XtNextEvent, XtProcessEvent, XtPeekEvent, XtPending, XtAddInput, XtAddTimeout, XtAddWorkProc, XtCreateApplicationShell, XtAddActions, XtSetSelectionTimeout, and XtGetSelectionTimeout.

```c
Widget XtInitialize(shell_name, application_class, options, num_options, argc, argv)
    String shell_name;
    String application_class;
    XrmOptionDescRec options[];
    Cardinal num_options;
    int *argc;
    String argv[];

    shell_name  This parameter is ignored; therefore, you can specify NULL.
    application_class Specifies the class name of this application.
    options Specifies how to parse the command line for any application-specific resources. The options argument is passed as a parameter to XrmParseCommand.
    num_options Specifies the number of entries in the options list.
    argc Specifies a pointer to the number of command line parameters.
    argv Specifies the command line parameters.
```
XtInitialize calls XtToolkitInitialize to initialize the toolkit internals, creates a default application context for use by the other convenience routines, calls XtOpenDisplay with display_string NULL and application_name NULL, and finally calls XtAppCreateShell with application_name NULL and returns the created shell. The semantics of calling XtInitialize more than once are undefined. This routine has been replaced by XtAppInitialize.

void XtMainLoop(void)

XtMainLoop first reads the next alternate input, timer, or X event by calling XtNextEvent. Then it dispatches this to the appropriate registered procedure by calling XtDispatchEvent. This routine has been replaced by XtAppMainLoop.

void XtNextEvent(event_return)

XEvent *event_return;

event_return Returns the event information to the specified event structure.

If no input is on the X input queue for the default application context, XtNextEvent flushes the X output buffer and waits for an event while looking at the alternate input sources and timeout values and calling any callback procedures triggered by them. This routine has been replaced by XtAppNextEvent. XtInitialize must be called before using this routine.

void XtProcessEvent(mask)

XtInputMask mask;

mask Specifies the type of input to process.

XtProcessEvent processes one X event, timeout, or alternate input source (depending on the value of mask), blocking if necessary. It has been replaced by XtAppProcessEvent. XtInitialize must be called before using this function.

Boolean XtPeekEvent(event_return)

XEvent *event_return;

event_return Returns the event information to the specified event structure.

If there is an event in the queue for the default application context, XtPeekEvent fills in the event and returns a nonzero value. If no X input is on the queue, XtPeekEvent flushes the output buffer and blocks until input is available, possibly calling some timeout callbacks in the process. If the input is an event, XtPeekEvent fills in the event and returns a nonzero value. Otherwise, the input is for an alternate input source, and XtPeekEvent returns zero. This routine has been replaced by XtAppPeekEvent. XtInitialize must be called before using this routine.

Boolean XtPending()

XtPending returns a nonzero value if there are events pending from the X server or alternate input sources in the default application context. If there are no events pending, it flushes the output buffer and returns a zero value. It has been replaced by XtAppPending. XtInitialize must be called before using this routine.

XtInputId XtAddInput(source, condition, proc, client_data)

int source;
XtPointer condition;
XtInputCallbackProc proc;
XtPointer client_data;
source Specifies the source file descriptor on a POSIX-based system or other operating-system-dependent device specification.

condition Specifies the mask that indicates either a read, write, or exception condition or some operating-system-dependent condition.

proc Specifies the procedure called when input is available.

client_data Specifies the parameter to be passed to proc when input is available.

The XtAddInput function registers in the default application context a new source of events, which is usually file input but can also be file output. (The word file should be loosely interpreted to mean any sink or source of data.) XtAddInput also specifies the conditions under which the source can generate events. When input is pending on this source in the default application context, the callback procedure is called. This routine has been replaced by XtAppAddInput. XtInitialize must be called before using this routine.

XtIntervalId XtAddTimeOut(interval, proc, client_data)

unsigned long interval;
XtTimerCallbackProc proc;
XtPointer client_data;

interval Specifies the time interval in milliseconds.
proc Specifies the procedure to be called when time expires.
client_data Specifies the parameter to be passed to proc when it is called.

The XtAddTimeOut function creates a timeout in the default application context and returns an identifier for it. The timeout value is set to interval. The callback procedure will be called after the time interval elapses, after which the timeout is removed. This routine has been replaced by XtAppAddTimeOut. XtInitialize must be called before using this routine.

XtWorkProcId XtAddWorkProc(proc, client_data)

XtWorkProc proc;
XtPointer client_data;

proc Procedure to call to do the work.
client_data Client data to pass to proc when it is called.

This routine registers a work procedure in the default application context. It has been replaced by XtAppAddWorkProc. XtInitialize must be called before using this routine.

Widget XtCreateApplicationShell(name, widget_class, args, num_args)

String name;
WidgetClass widget_class;
ArgList args;
Cardinal num_args;

name This parameter is ignored; therefore, you can specify NULL.
widget_class Specifies the widget class pointer for the created application shell widget. This will usually be topLevelShellWidgetClass or a subclass thereof.
args Specifies the argument list to override any other resource specifications.
num_args Specifies the number of entries in args.

The procedure XtCreateApplicationShell calls XtAppCreateShell with application_name NULL, the application class passed to XtInitialize, and the default application context created by XtInitialize. This routine has been replaced by XtAppCreateShell.
An old-format resource type converter procedure pointer is of type `XtConverter`.

```c
typedef void (*XtConverter)(XrmValue*, Cardinal*, XrmValue*, XrmValue*);
```

- `args` Specifies a list of additional `XrmValue` arguments to the converter if additional context is needed to perform the conversion, or NULL.
- `num_args` Specifies the number of entries in `args`.
- `from` Specifies the value to convert.
- `to` Specifies the descriptor to use to return the converted value.

Type converters should perform the following actions:
- Check to see that the number of arguments passed is correct.
- Attempt the type conversion.
- If successful, return the size and pointer to the data in the `to` argument; otherwise, call `XtWarningMsg` and return without modifying the `to` argument.

Most type converters just take the data described by the specified `from` argument and return data by writing into the specified `to` argument. A few need other information, which is available in the specified argument list. A type converter can invoke another type converter, which allows differing sources that may convert into a common intermediate result to make maximum use of the type converter cache.

Note that the address returned in `to->addr` cannot be that of a local variable of the converter because this is not valid after the converter returns. It should be a pointer to a static variable.

The procedure type `XtConverter` has been replaced by `XtTypeConverter`.

The `XtStringConversionWarning` function is a convenience routine for old-format resource converters that convert from strings.

```c
void XtStringConversionWarning(char* src, const char* dst_type)
```

- `src` Specifies the string that could not be converted.
- `dst_type` Specifies the name of the type to which the string could not be converted.

The `XtStringConversionWarning` function issues a warning message with name "conversionError", type "string", class "XtToolkitError", and the default message string "Cannot convert "src" to type dst_type". This routine has been superseded by `XtDisplayStringConversionWarning`.

To register an old-format converter, use `XtAddConverter` or `XtAppAddConverter`.

```c
void XtAddConverter(String from_type, String to_type, XtConverter converter, XtConvertArgList convert_args, Cardinal num_args)
```

- `from_type` Specifies the source type.
- `to_type` Specifies the destination type.
 converter  Specifies the type converter procedure.
 convert_args  Specifies how to compute the additional arguments to the converter, or NULL.
 num_args  Specifies the number of entries in convert_args.

`XtAddConverter` is equivalent in function to `XtSetTypeConverter` with `cache_type` equal to `XtCacheAll` for old-format type converters. It has been superseded by `XtSetTypeConverter`.

```c
void XtAppAddConverter(app_context, from_type, to_type, converter, convert_args, num_args)
    XtAppContext app_context;
    String from_type;
    String to_type;
    XtConverter converter;
    XtConvertArgList convert_args;
    Cardinal num_args;

    app_context  Specifies the application context.
    from_type  Specifies the source type.
    to_type  Specifies the destination type.
    converter  Specifies the type converter procedure.
    convert_args  Specifies how to compute the additional arguments to the converter, or NULL.
    num_args  Specifies the number of entries in convert_args.

XtAppAddConverter` is equivalent in function to `XtAppSetTypeConverter` with `cache_type` equal to `XtCacheAll` for old-format type converters. It has been superseded by `XtAppSetTypeConverter`.

To invoke resource conversions, a client may use `XtConvert` or, for old-format converters only, `XtDirectConvert`.

```c
void XtConvert(w, from_type, from, to_type, to_return)
    Widget w;
    String from_type;
    XrmValuePtr from;
    String to_type;
    XrmValuePtr to_return;

w  Specifies the widget to use for additional arguments, if any are needed. 01
from_type  Specifies the source type.
from  Specifies the value to be converted.

void XtDirectConvert(converter, args, num_args, from, to_return)
    XtConverter converter;
    XrmValuePtr args;
    Cardinal num_args;
    XrmValuePtr from;
    XrmValuePtr to_return;
```
The `XtConvert` function looks up the type converter registered to convert `from_type` to `to_type`, computes any additional arguments needed, and then calls `XtDirectConvert` or `XtCallConverter`. The `XtDirectConvert` function looks in the converter cache to see if this conversion procedure has been called with the specified arguments. If so, it returns a descriptor for information stored in the cache; otherwise, it calls the converter and enters the result in the cache.

Before calling the specified converter, `XtDirectConvert` sets the return value size to zero and the return value address to NULL. To determine if the conversion was successful, the client should check `to_return.addr` for non-NULL. The data returned by `XtConvert` must be copied immediately by the caller, as it may point to static data in the type converter.

`XtConvert` has been replaced by `XtConvertAndStore`, and `XtDirectConvert` has been superseded by `XtCallConverter`.

To deallocate a shared GC when it is no longer needed, use `XtDestroyGC`.

```c
void XtDestroyGC(w, gc)
    Widget w;
    GC gc;

w    Specifies any object on the display for which the shared GC was created.
    Must be of class Object or any subclass thereof.

gc   Specifies the shared GC to be deallocated.
```

References to sharable GCs are counted and a free request is generated to the server when the last user of a given GC destroys it. Note that some earlier versions of `XtDestroyGC` had only a `gc` argument. Therefore, this function is not very portable, and you are encouraged to use `XtReleaseGC` instead.

To declare an action table in the default application context and register it with the translation manager, use `XtAddActions`.

```c
void XtAddActions(actions, num_actions)
    XtActionList actions;
    Cardinal num_actions;

actions   Specifies the action table to register.
num_actions Specifies the number of entries in `actions`.
```

If more than one action is registered with the same name, the most recently registered action is used. If duplicate actions exist in an action table, the first is used. The Intrinsics register an action table for `XtMenuPopup` and `XtMenuPopdown` as part of X Toolkit initialization. This routine has been replaced by `XtAppAddActions`. `XtInitialize` must be called before using this routine.

To set the Intrinsics selection timeout in the default application context, use `XtSetSelectionTimeout`.

```c
200
```
void XtSetSelectionTimeout(timeout)
    unsigned long timeout;

  timeout Specifies the selection timeout in milliseconds. This routine has been replaced by XtAppSetSelectionTimeout. XtInitialize must be called before using this routine.

To get the current selection timeout value in the default application context, use XtGetSelectionTimeout.

unsigned long XtGetSelectionTimeout()

The selection timeout is the time within which the two communicating applications must respond to one another. If one of them does not respond within this interval, the Intrinsics abort the selection request.

This routine has been replaced by XtAppGetSelectionTimeout. XtInitialize must be called before using this routine.

To get the current selection timeout value in the default application context, use XtGetSelectionTimeout.

unsigned long XtGetSelectionTimeout()

The selection timeout is the time within which the two communicating applications must respond to one another. If one of them does not respond within this interval, the Intrinsics abort the selection request.

This routine has been replaced by XtAppGetSelectionTimeout. XtInitialize must be called before using this routine.

To get the current selection timeout value in the default application context, use XtGetSelectionTimeout.

unsigned long XtGetSelectionTimeout()

The selection timeout is the time within which the two communicating applications must respond to one another. If one of them does not respond within this interval, the Intrinsics abort the selection request.

This routine has been replaced by XtAppGetSelectionTimeout. XtInitialize must be called before using this routine.

To obtain the global error database (for example, to merge with an application- or widget-specific database), use XtGetErrorDatabase.

XrmDatabase *XtGetErrorDatabase()

The XtGetErrorDatabase function returns the address of the error database. The Intrinsics do a lazy binding of the error database and do not merge in the database file until the first call to XtGetErrorDatabaseText. This routine has been replaced by XtAppGetErrorDatabase.

An error message handler can obtain the error database text for an error or a warning by calling XtGetErrorDatabaseText.

void XtGetErrorDatabaseText(name, type, class, default, buffer_return, nbytes)
    String name, type, class;
    String default;
    String buffer_return;
    int nbytes;

name Specify the name and type that are concatenated to form the resource name of the error message.

type Specifies the resource class of the error message.

class Specifies the default message to use if an error database entry is not found.

default buffer_return Specifies the buffer into which the error message is to be returned.

buffer_return nbytes Specifies the size of the buffer in bytes.

The XtGetErrorDatabaseText returns the appropriate message from the error database associated with the default application context or returns the specified default message if one is not found in the error database. To form the full resource name and class when querying the database, the name and type are concatenated with a single "." between them and the class is concatenated with itself with a single "." if it does not already contain a ".". This routine has been superseded by XtAppGetErrorDatabaseText.

To register a procedure to be called on fatal error conditions, use XtSetErrorMsgHandler.

void XtSetErrorMsgHandler(msg_handler)
    XtErrorMsgHandler msg_handler;

201
msg_handler  Specifies the new fatal error procedure, which should not return.
The default error handler provided by the Intrinsics constructs a string from the error resource
database and calls XtError. Fatal error message handlers should not return. If one does,
subsequent Intrinsics behavior is undefined. This routine has been superseded by
XtAppSetErrorMsgHandler.

To call the high-level error handler, use XtErrorMsg.

void XtErrorMsg(name, type, class, default, params, num_params)
  String name;
  String type;
  String class;
  String default;
  String *params;
  Cardinal *num_params;

name  Specifies the general kind of error.
type  Specifies the detailed name of the error.
class  Specifies the resource class.
default  Specifies the default message to use if an error database entry is not found.
params  Specifies a pointer to a list of values to be stored in the message.
num_params  Specifies the number of entries in params.

This routine has been superseded by XtAppErrorMsg.

To register a procedure to be called on nonfatal error conditions, use XtSetWarningMsgHandler.

void XtSetWarningMsgHandler(msg_handler)
  XtErrorMsgHandler msg_handler;

msg_handler  Specifies the new nonfatal error procedure, which usually returns.
The default warning handler provided by the Intrinsics constructs a string from the error
resource database and calls XtWarning. This routine has been superseded by
XtAppSetWarningMsgHandler.

To call the installed high-level warning handler, use XtWarningMsg.

void XtWarningMsg(name, type, class, default, params, num_params)
  String name;
  String type;
  String class;
  String default;
  String *params;
  Cardinal *num_params;

name  Specifies the general kind of error.
type  Specifies the detailed name of the error.
class  Specifies the resource class.
default  Specifies the default message to use if an error database entry is not found.
params  Specifies a pointer to a list of values to be stored in the message.
num_params  Specifies the number of entries in params.

This routine has been superseded by XtAppWarningMsg.
To register a procedure to be called on fatal error conditions, use `XtSetErrorHandler`.

```c
void XtSetErrorHandler(handler)
    XtErrorHandler handler;
```

`handler` specifies the new fatal error procedure, which should not return.

The default error handler provided by the Intrinsics is `XtError`. On POSIX-based systems, it prints the message to standard error and terminates the application. Fatal error message handlers should not return. If one does, subsequent X Toolkit behavior is undefined. This routine has been superseded by `XtAppSetErrorHandler`.

To call the installed fatal error procedure, use `XtError`.

```c
void XtError(message)
    String message;
```

`message` specifies the message to be reported.

Most programs should use `XtAppErrorMsg`, not `XtError`, to provide for customization and internationalization of error messages. This routine has been superseded by `XtAppError`.

To register a procedure to be called on nonfatal error conditions, use `XtSetWarningHandler`.

```c
void XtSetWarningHandler(handler)
    XtErrorHandler handler;
```

`handler` specifies the new nonfatal error procedure, which usually returns.

The default warning handler provided by the Intrinsics is `XtWarning`. On POSIX-based systems, it prints the message to standard error and returns to the caller. This routine has been superseded by `XtAppSetWarningHandler`.

To call the installed nonfatal error procedure, use `XtWarning`.

```c
void XtWarning(message)
    String message;
```

`message` specifies the nonfatal error message to be reported.

Most programs should use `XtAppWarningMsg`, not `XtWarning`, to provide for customization and internationalization of warning messages. This routine has been superseded by `XtAppWarning`.

203
Appendix D

Intrinsics Error Messages

All Intrinsics errors and warnings have class "XtToolkitError". The following two tables summarize the common errors and warnings that can be generated by the Intrinsics. Additional implementation-dependent messages are permitted.

## Error Messages

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocError</td>
<td>calloc</td>
<td>Cannot perform calloc</td>
</tr>
<tr>
<td>allocError</td>
<td>malloc</td>
<td>Cannot perform malloc</td>
</tr>
<tr>
<td>allocError</td>
<td>realloc</td>
<td>Cannot perform realloc</td>
</tr>
<tr>
<td>communicationError</td>
<td>select</td>
<td>Select failed</td>
</tr>
<tr>
<td>internalError</td>
<td>shell</td>
<td>Shell's window manager interaction is broken</td>
</tr>
<tr>
<td>invalidArgCount</td>
<td>xtGetValues</td>
<td>Argument count &gt; 0 on NULL argument list in XtGetValues</td>
</tr>
<tr>
<td>invalidArgCount</td>
<td>xtSetValues</td>
<td>Argument count &gt; 0 on NULL argument list in XtSetValues</td>
</tr>
<tr>
<td>invalidClass</td>
<td>constraintSetValue</td>
<td>Subclass of Constraint required in CallConstraintSetValue</td>
</tr>
<tr>
<td>invalidClass</td>
<td>XtAppCreateShell</td>
<td>XtAppCreateShell requires non-NULL widget class</td>
</tr>
<tr>
<td>invalidClass</td>
<td>XtCreatePopupShell</td>
<td>XtCreatePopupShell requires non-NULL widget class</td>
</tr>
<tr>
<td>invalidClass</td>
<td>XtCreateWidget</td>
<td>XtCreateWidget requires non-NULL widget class</td>
</tr>
<tr>
<td>invalidClass</td>
<td>xtPopdown</td>
<td>XtPopdown requires a subclass of shellWidgetClass</td>
</tr>
<tr>
<td>invalidClass</td>
<td>XtPopup</td>
<td>XtPopup requires a subclass of shellWidgetClass</td>
</tr>
<tr>
<td>invalidDimension</td>
<td>XtCreateWindow</td>
<td>Widget %s has zero width and/or height</td>
</tr>
<tr>
<td>invalidDimension</td>
<td>shellRealize</td>
<td>Shell widget %s has zero width and/or height</td>
</tr>
<tr>
<td>invalidDisplay</td>
<td>xtInitialize</td>
<td>Can't Open display</td>
</tr>
<tr>
<td>invalidGeometryManager</td>
<td>xtMakeGeometryRequest</td>
<td>XtMakeGeometryRequest - parent has no geometry manager</td>
</tr>
<tr>
<td>invalidParameter</td>
<td>removePopupFromParent</td>
<td>RemovePopupFromParent requires non-NULL popuplist invalid condition passed to XtAddInput</td>
</tr>
<tr>
<td>invalidParameter</td>
<td>XtAddInput</td>
<td></td>
</tr>
<tr>
<td>invalidParameters</td>
<td>xMenuPopupAction</td>
<td>MenuPopup wants exactly one argument</td>
</tr>
<tr>
<td>invalidParent</td>
<td>realize</td>
<td>Application shell is not a windowed widget?</td>
</tr>
<tr>
<td>invalidParent</td>
<td>XtCreatePopupShell</td>
<td>XtCreatePopupShell requires non-NULL parent</td>
</tr>
<tr>
<td>invalidParent</td>
<td>XtCreateWidget</td>
<td>XtCreateWidget requires non-NULL parent</td>
</tr>
<tr>
<td>invalidParent</td>
<td>xtMakeGeometryRequest</td>
<td>XtMakeGeometryRequest - NULL parent. Use Set-Values instead</td>
</tr>
<tr>
<td>invalidParent</td>
<td>xtManageChildren</td>
<td>Attempt to manage a child when parent is not Composite</td>
</tr>
<tr>
<td>invalidParent</td>
<td>xtUnmanageChildren</td>
<td>Attempt to unmanage a child when parent is not Composite</td>
</tr>
<tr>
<td>invalidProcedure</td>
<td>inheritanceProc</td>
<td>Unresolved inheritance operation</td>
</tr>
<tr>
<td>invalidProcedure</td>
<td>realizeProc</td>
<td>No realize class procedure defined</td>
</tr>
<tr>
<td>invalidWindow</td>
<td>eventHandler</td>
<td>Event with wrong window</td>
</tr>
</tbody>
</table>

204
Events are disappearing from under Shell
Could’t find ancestor with display information

Could’t find per display information

internal error: no selection property context for display
NULL insert_child procedure

Widget class %s found when subclass of %s expected: %s

Trying to merge translation tables with cycles, and can’t resolve this cycle.

---

**Warning Messages**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambiguousParent</td>
<td>xtManageChildren</td>
<td>Not all children have same parent in XtManageChildren</td>
</tr>
<tr>
<td>ambiguousParent</td>
<td>xtUnmanageChildren</td>
<td>Not all children have same parent in XtUnmanageChildren</td>
</tr>
<tr>
<td>communicationError</td>
<td>windowManager</td>
<td>Window Manager is confused</td>
</tr>
<tr>
<td>conversionError</td>
<td>string</td>
<td>Cannot convert string “%s” to type %s</td>
</tr>
<tr>
<td>displayError</td>
<td>invalidDisplay</td>
<td>Can’t find display structure</td>
</tr>
<tr>
<td>grabError</td>
<td>xtAddGrab</td>
<td>XtAddGrab requires exclusive grab if spring_loaded is TRUE</td>
</tr>
<tr>
<td>grabError</td>
<td>grabDestroyCallback</td>
<td>XtAddGrab requires exclusive grab if spring_loaded is TRUE</td>
</tr>
<tr>
<td>grabError</td>
<td>xtRemoveGrab</td>
<td>XtRemoveGrab asked to remove a widget not on the grab list</td>
</tr>
<tr>
<td>initializationError</td>
<td>xtInitialize</td>
<td>Initializing Resource Lists twice</td>
</tr>
<tr>
<td>invalidArgCount</td>
<td>getResources</td>
<td>argument count &gt; 0 on NULL argument list</td>
</tr>
<tr>
<td>invalidCallbackList</td>
<td>xtAddCallbacks</td>
<td>Cannot find callback list in XtAddCallbacks</td>
</tr>
<tr>
<td>invalidCallbackList</td>
<td>xtCallCallback</td>
<td>Cannot find callback list in XtCallCallback</td>
</tr>
<tr>
<td>invalidCallbackList</td>
<td>xtOverrideCallback</td>
<td>Cannot find callback list in XtOverrideCallback</td>
</tr>
<tr>
<td>invalidCallbackList</td>
<td>xtRemoveAllCallback</td>
<td>Cannot find callback list in XtRemoveAllCallback</td>
</tr>
<tr>
<td>invalidCallbackList</td>
<td>xtRemoveCallbacks</td>
<td>Cannot find callback list in XtRemoveCallbacks</td>
</tr>
<tr>
<td>invalidChild</td>
<td>xtManageChildren</td>
<td>null child passed to XtManageChildren</td>
</tr>
<tr>
<td>invalidChild</td>
<td>xtUnmanageChildren</td>
<td>Null child passed to XtUnmanageChildren</td>
</tr>
<tr>
<td>invalidDepth</td>
<td>setValues</td>
<td>Can’t change widget depth</td>
</tr>
<tr>
<td>invalidGeometry</td>
<td>xtMakeGeometryRequest</td>
<td>Shell subclass did not take care of geometry in XtSetValues</td>
</tr>
<tr>
<td>invalidParameters</td>
<td>compileAccelerators</td>
<td>String to AcceleratorTable needs no extra arguments</td>
</tr>
<tr>
<td>invalidParameters</td>
<td>compileTranslations</td>
<td>String to TranslationTable needs no extra arguments</td>
</tr>
<tr>
<td>invalidParameters</td>
<td>mergeTranslations</td>
<td>MergeTM to TranslationTable needs no extra arguments</td>
</tr>
<tr>
<td>invalidParameters</td>
<td>xtMenuPopdown</td>
<td>XtMenuPopdown called with num_params != 0 or 1</td>
</tr>
<tr>
<td>invalidParent</td>
<td>xtCopyFromParent</td>
<td>CopyFromParent must have non-NUL parent</td>
</tr>
<tr>
<td>invalidPopup</td>
<td>xtMenuPopup</td>
<td>Can’t find popup in XtMenuPopup</td>
</tr>
<tr>
<td>invalidPopup</td>
<td>xtMenuPopdown</td>
<td>Can’t find popup in XtMenuPopdown</td>
</tr>
<tr>
<td>invalidPopup</td>
<td>unsupportedOperation</td>
<td>Pop-up menu creation is only supported on ButtonPress or EnterNotify events.</td>
</tr>
<tr>
<td>invalidPopup</td>
<td>unsupportedOperation</td>
<td>Pop-up menu creation is only supported on ButtonPress or EnterNotify events.</td>
</tr>
<tr>
<td>invalidProcedure</td>
<td>deleteChild</td>
<td>null delete_child procedure in XtDestroy</td>
</tr>
<tr>
<td>Error Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>invalidProcedure</td>
<td>Invalid procedure should not be NULL</td>
<td></td>
</tr>
<tr>
<td>invalidResourceCount</td>
<td>Resource count &gt; 0 on NULL resource list</td>
<td></td>
</tr>
<tr>
<td>invalidResourceName</td>
<td>Cannot find resource name %s as argument to conversion</td>
<td></td>
</tr>
<tr>
<td>invalidShell</td>
<td>Widget has no shell ancestor</td>
<td></td>
</tr>
<tr>
<td>invalidSizeOverride</td>
<td>Representation size %d must match superclass's to override %s</td>
<td></td>
</tr>
<tr>
<td>invalidTypeOverride</td>
<td>Representation type %s must match superclass's to override %s</td>
<td></td>
</tr>
<tr>
<td>invalidWidget</td>
<td>RemovePopupFromParent, widget not on parent list</td>
<td></td>
</tr>
<tr>
<td>missingCharsetList</td>
<td>Missing charsets in String to FontSet conversion</td>
<td></td>
</tr>
<tr>
<td>noColormap</td>
<td>Cannot allocate colormap entry for &quot;%s&quot;</td>
<td></td>
</tr>
<tr>
<td>registerWindowError</td>
<td>Attempt to change already registered window.</td>
<td></td>
</tr>
<tr>
<td>registerWindowError</td>
<td>Attempt to unregister invalid window.</td>
<td></td>
</tr>
<tr>
<td>translation error</td>
<td>Can't remove accelerators from NULL table</td>
<td></td>
</tr>
<tr>
<td>translation error</td>
<td>Tried to remove non-existant accelerators</td>
<td></td>
</tr>
<tr>
<td>translationError</td>
<td>Overriding earlier translation manager actions.</td>
<td></td>
</tr>
<tr>
<td>translationError</td>
<td>Old translation table was null, cannot modify.</td>
<td></td>
</tr>
<tr>
<td>translationError</td>
<td>Can't translate event through NULL table</td>
<td></td>
</tr>
<tr>
<td>translationError</td>
<td>Actions not found: %s</td>
<td></td>
</tr>
<tr>
<td>translationError</td>
<td>Initializing Translation manager twice.</td>
<td></td>
</tr>
<tr>
<td>translationError</td>
<td>... found while parsing &quot;%s&quot;</td>
<td></td>
</tr>
<tr>
<td>translationTableSyntaxError</td>
<td>translation table syntax error: %s</td>
<td></td>
</tr>
<tr>
<td>typeConversionError</td>
<td>No type converter registered for ' %s ' to ' %s ' conversion.</td>
<td></td>
</tr>
<tr>
<td>versionMismatch</td>
<td>Widget class %s version mismatch: widget %d vs. intrinsics %d.</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>Pixel to color conversion needs screen and colormap arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>Integer to Bool conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>Integer to Boolean conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>Integer to Font conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>Integer to Pixel conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>Integer toPixmap conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>Integer to Short conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to Bool conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to Boolean conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to Cursor conversion needs screen argument</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to Display conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to File conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to Font conversion needs screen argument</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to FontSet conversion needs display and locale arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to cursor conversion needs screen argument</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to Integer conversion needs no extra arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>String to pixel conversion needs screen and colormap arguments</td>
<td></td>
</tr>
<tr>
<td>wrongParameters</td>
<td>Color to Pixe conversion needs no extra arguments</td>
<td></td>
</tr>
</tbody>
</table>

Pixel to color conversion needs screen and colormap arguments.
# Appendix E

## Defined Strings

The `StringDefs.h` header file contains definitions for the following resource name, class, and representation type symbolic constants.

### Resource names:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNaccelerators</td>
<td>&quot;accelerators&quot;</td>
</tr>
<tr>
<td>XtNallowHoriz</td>
<td>&quot;allowHoriz&quot;</td>
</tr>
<tr>
<td>XtNallowVert</td>
<td>&quot;allowVert&quot;</td>
</tr>
<tr>
<td>XtNancestorSensitive</td>
<td>&quot;ancestorSensitive&quot;</td>
</tr>
<tr>
<td>XtNbackground</td>
<td>&quot;background&quot;</td>
</tr>
<tr>
<td>XtNbackgroundPixmap</td>
<td>&quot;backgroundPixmap&quot;</td>
</tr>
<tr>
<td>XtNbitmap</td>
<td>&quot;bitmap&quot;</td>
</tr>
<tr>
<td>XtNborderColor</td>
<td>&quot;borderColor&quot;</td>
</tr>
<tr>
<td>XtNborder</td>
<td>&quot;borderWidth&quot;</td>
</tr>
<tr>
<td>XtNborderPixmap</td>
<td>&quot;borderPixmap&quot;</td>
</tr>
<tr>
<td>XtNborderWidth</td>
<td>&quot;borderWidth&quot;</td>
</tr>
<tr>
<td>XtNcallback</td>
<td>&quot;callback&quot;</td>
</tr>
<tr>
<td>XtNchildren</td>
<td>&quot;children&quot;</td>
</tr>
<tr>
<td>XtNcolormap</td>
<td>&quot;colormap&quot;</td>
</tr>
<tr>
<td>XtNdepth</td>
<td>&quot;depth&quot;</td>
</tr>
<tr>
<td>XtNdestroyCallback</td>
<td>&quot;destroyCallback&quot;</td>
</tr>
<tr>
<td>XtNeditType</td>
<td>&quot;editType&quot;</td>
</tr>
<tr>
<td>XtNfile</td>
<td>&quot;file&quot;</td>
</tr>
<tr>
<td>XtNfont</td>
<td>&quot;font&quot;</td>
</tr>
<tr>
<td>XtNfontSet</td>
<td>&quot;fontSet&quot;</td>
</tr>
<tr>
<td>XtNforceBars</td>
<td>&quot;forceBars&quot;</td>
</tr>
<tr>
<td>XtNforeground</td>
<td>&quot;foreground&quot;</td>
</tr>
<tr>
<td>XtNfunction</td>
<td>&quot;function&quot;</td>
</tr>
<tr>
<td>XtNheight</td>
<td>&quot;height&quot;</td>
</tr>
<tr>
<td>XtNhighlight</td>
<td>&quot;highlight&quot;</td>
</tr>
<tr>
<td>XtNhSpace</td>
<td>&quot;hSpace&quot;</td>
</tr>
<tr>
<td>XtNindex</td>
<td>&quot;index&quot;</td>
</tr>
<tr>
<td>XtNinitialResourcesPersistent</td>
<td>&quot;initialResourcesPersistent&quot;</td>
</tr>
<tr>
<td>XtNinnerHeight</td>
<td>&quot;innerHeight&quot;</td>
</tr>
<tr>
<td>XtNinnerWidth</td>
<td>&quot;innerWidth&quot;</td>
</tr>
<tr>
<td>XtNinnerWindow</td>
<td>&quot;innerWindow&quot;</td>
</tr>
<tr>
<td>XtNinsertPosition</td>
<td>&quot;insertPosition&quot;</td>
</tr>
<tr>
<td>XtNinternalHeight</td>
<td>&quot;internalHeight&quot;</td>
</tr>
<tr>
<td>XtNinternalWidth</td>
<td>&quot;internalWidth&quot;</td>
</tr>
<tr>
<td>XtNjumpProc</td>
<td>&quot;jumpProc&quot;</td>
</tr>
<tr>
<td>XtNjustify</td>
<td>&quot;justify&quot;</td>
</tr>
<tr>
<td>XtNknobHeight</td>
<td>&quot;knobHeight&quot;</td>
</tr>
<tr>
<td>XtNknobIndent</td>
<td>&quot;knobIndent&quot;</td>
</tr>
<tr>
<td>XtNknobPixel</td>
<td>&quot;knobPixel&quot;</td>
</tr>
<tr>
<td>XtNknobWidth</td>
<td>&quot;knobWidth&quot;</td>
</tr>
<tr>
<td>Symbol</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>XtNAccelerators</td>
<td>&quot;Accelerators&quot;</td>
</tr>
<tr>
<td>XtCBackgroundColor</td>
<td>&quot;Background&quot;</td>
</tr>
</tbody>
</table>

208
X Toolkit Intrinsics

XtCBitmap
XtCBoolean
XtCBorderColor
XtCBorderWidth
XtCCallback
XtCColormap
XtCColor
XtCCursor
XtCDepth
XtCEditType
XtCEventBindings
XtCFile
XtCFont
XtCFontSet
XtCForeground
XtCFraction
XtCFunctlona
XtCHeight
XtCHSpace
XtCIndex
XtCInitialResourcesPersistent
XtCInsertPosition
XtCInterval
XtCJustify
XtCKnobIndent
XtCKnobPixel
XtCLabel
XtCLength
XtCMappedWhenManaged
XtCMargin
XtCMenuEntry
XtCNotify
XtCOrientation
XtCParameter
XtCPixmap
XtCPosition
XtCReadOnly
XtCResize
XtCReverseVideo
XtCScreen
XtCScrollProc
XtCScrollDCursor
XtCScrollHCursor
XtCScrollLCursor
XtCScrollRCursor
XtCScrollUCursor
XtCScrollVCursor
XtCSelection
XtCSensitive
XtCSelectionArray
XtCSpace
XtCString
XtCTextOptions
XtCTextPosition
X Toolkit Intrinsics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtCTextSink</td>
<td>&quot;TextSink&quot;</td>
</tr>
<tr>
<td>XtCTextSource</td>
<td>&quot;TextSource&quot;</td>
</tr>
<tr>
<td>XtCThickness</td>
<td>&quot;Thickness&quot;</td>
</tr>
<tr>
<td>XtCThumb</td>
<td>&quot;Thumb&quot;</td>
</tr>
<tr>
<td>XtCTranslations</td>
<td>&quot;Translations&quot;</td>
</tr>
<tr>
<td>XtCValue</td>
<td>&quot;Value&quot;</td>
</tr>
<tr>
<td>XtCVSpace</td>
<td>&quot;VSpace&quot;</td>
</tr>
<tr>
<td>XtCWidth</td>
<td>&quot;Width&quot;</td>
</tr>
<tr>
<td>XtCWindow</td>
<td>&quot;Window&quot;</td>
</tr>
<tr>
<td>XtCX</td>
<td>&quot;X&quot;</td>
</tr>
<tr>
<td>XtCY</td>
<td>&quot;Y&quot;</td>
</tr>
</tbody>
</table>

Resource representation types:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtRAcceleratorTable</td>
<td>&quot;AcceleratorTable&quot;</td>
</tr>
<tr>
<td>XtRAtom</td>
<td>&quot;Atom&quot;</td>
</tr>
<tr>
<td>XtRBitmap</td>
<td>&quot;Bitmap&quot;</td>
</tr>
<tr>
<td>XtRBool</td>
<td>&quot;Bool&quot;</td>
</tr>
<tr>
<td>XtRBoolean</td>
<td>&quot;Boolean&quot;</td>
</tr>
<tr>
<td>XtRCallback</td>
<td>&quot;Callback&quot;</td>
</tr>
<tr>
<td>XtRCallProc</td>
<td>&quot;CallProc&quot;</td>
</tr>
<tr>
<td>XtRCardinal</td>
<td>&quot;Cardinal&quot;</td>
</tr>
<tr>
<td>XtRColor</td>
<td>&quot;Color&quot;</td>
</tr>
<tr>
<td>XtRColormap</td>
<td>&quot;Colormap&quot;</td>
</tr>
<tr>
<td>XtRCursor</td>
<td>&quot;Cursor&quot;</td>
</tr>
<tr>
<td>XtRDimension</td>
<td>&quot;Dimension&quot;</td>
</tr>
<tr>
<td>XtRDisplay</td>
<td>&quot;Display&quot;</td>
</tr>
<tr>
<td>XtREditMode</td>
<td>&quot;EditMode&quot;</td>
</tr>
<tr>
<td>XtREnum</td>
<td>&quot;Enum&quot;</td>
</tr>
<tr>
<td>XtRFile</td>
<td>&quot;File&quot;</td>
</tr>
<tr>
<td>XtRFloat</td>
<td>&quot;Float&quot;</td>
</tr>
<tr>
<td>XtRFont</td>
<td>&quot;Font&quot;</td>
</tr>
<tr>
<td>XtRFontSet</td>
<td>&quot;FontSet&quot;</td>
</tr>
<tr>
<td>XtRFontStruct</td>
<td>&quot;FontStruct&quot;</td>
</tr>
<tr>
<td>XtRFunction</td>
<td>&quot;Function&quot;</td>
</tr>
<tr>
<td>XtRGeometry</td>
<td>&quot;Geometry&quot;</td>
</tr>
<tr>
<td>XtRImmediate</td>
<td>&quot;Immediate&quot;</td>
</tr>
<tr>
<td>XtRInitialState</td>
<td>&quot;InitialState&quot;</td>
</tr>
<tr>
<td>XtRInt</td>
<td>&quot;Int&quot;</td>
</tr>
<tr>
<td>XtRJustify</td>
<td>&quot;Justify&quot;</td>
</tr>
<tr>
<td>XtRLongBoolean</td>
<td>XtRBool</td>
</tr>
<tr>
<td>XtRObject</td>
<td>&quot;Object&quot;</td>
</tr>
<tr>
<td>XtROrientation</td>
<td>&quot;Orientation&quot;</td>
</tr>
<tr>
<td>XtRPixel</td>
<td>&quot;Pixel&quot;</td>
</tr>
<tr>
<td>XtRPixmap</td>
<td>&quot;Pixmap&quot;</td>
</tr>
<tr>
<td>XtRPointer</td>
<td>&quot;Pointer&quot;</td>
</tr>
<tr>
<td>XtRPosition</td>
<td>&quot;Position&quot;</td>
</tr>
<tr>
<td>XtRScreen</td>
<td>&quot;Screen&quot;</td>
</tr>
<tr>
<td>XtRShort</td>
<td>&quot;Short&quot;</td>
</tr>
<tr>
<td>XtRString</td>
<td>&quot;String&quot;</td>
</tr>
<tr>
<td>XtRStringArray</td>
<td>&quot;StringArray&quot;</td>
</tr>
<tr>
<td>XtRStringTable</td>
<td>&quot;StringTable&quot;</td>
</tr>
</tbody>
</table>
Boolean enumeration constants:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtEoff</td>
<td>&quot;off&quot;</td>
</tr>
<tr>
<td>XtEfalse</td>
<td>&quot;false&quot;</td>
</tr>
<tr>
<td>XtEno</td>
<td>&quot;no&quot;</td>
</tr>
<tr>
<td>XtEon</td>
<td>&quot;on&quot;</td>
</tr>
<tr>
<td>XtEtrue</td>
<td>&quot;true&quot;</td>
</tr>
<tr>
<td>XtEyes</td>
<td>&quot;yes&quot;</td>
</tr>
</tbody>
</table>

Orientation enumeration constants:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtEvertical</td>
<td>&quot;vertical&quot;</td>
</tr>
<tr>
<td>XtEhorizontal</td>
<td>&quot;horizontal&quot;</td>
</tr>
</tbody>
</table>

Text edit enumeration constants:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtEtextRead</td>
<td>&quot;read&quot;</td>
</tr>
<tr>
<td>XtEtextAppend</td>
<td>&quot;append&quot;</td>
</tr>
<tr>
<td>XtEtextEdit</td>
<td>&quot;edit&quot;</td>
</tr>
</tbody>
</table>

Color enumeration constants:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtExtdefaultbackground</td>
<td>&quot;xtdefaultbackground&quot;</td>
</tr>
<tr>
<td>XtExtdefaultforeground</td>
<td>&quot;xtdefaultforeground&quot;</td>
</tr>
</tbody>
</table>

Font constant:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtExtdefaultfont</td>
<td>&quot;xtdefaultfont&quot;</td>
</tr>
</tbody>
</table>
Index

XtVaCreateWidget, 35

ButtonReleaseMask, 73

C

calloc, 150
Cardinal, 9, 35, 186
CenterGravity, 45
Chaining, 41, 42, 112
  Subclass, 18
  superclass, 18
changeManaged procedure, 53
CirculateNotify, 190, 193
CirculateRequest, 190, 193
Class Initialization, 19
classInitialize procedure, 19, 113
class_name, 14
ClientMessage, 100, 101, 102, 103, 190, 192, 193
colorConvertArgs, 115, 117, 124
ColormapNotify, 190, 193
compositeWidgetClass, 48
Composite widgets, 51
Composite, 6
CompositeClassExtension, 6, 52, 176, 179, 184
CompositeClassExtensionRec, 6
CompositeClassPart, 6, 23
compositeClassRec, 7
Composite.h, 21
CompositePart, 6, 7, 10, 44, 184
CompositeWidget, 6, 7
  Resources, 7
CompositeWidgetClass, 6
compositeWidgetClass, 6, 24, 37, 43, 45, 48, 49, 51
CompositeWidgetClass, 52
compositeWidgetClass, 53, 55, 56, 58, 76
compressEnterleave field, 97
compressExpose field, 97
compressMotion field, 96
ConfigureWindow, 75
ConfigureNotify, 47, 52, 190, 193
ConfigureRequest, 190, 193
Constraint.h, 14
constraintWidgetClass, 48, 49
Constraint.h, 12
Constraint, 8
  get_values_hook, 57
ConstraintClassExtension, 8, 57, 128
ConstraintClassExtensionRec, 8, 19
ConstraintClassPart, 8, 19, 23, 37, 42, 48, 49, 57, 108
constraintClassRec, 9
ConstraintPart, 8, 9, 133

A

Above, 77
Accelerator, 141
accept_focus procedure, 92
Action Table, 136
actions, 136
action_proc procedure, 135
AnyButton, 89
AnyKey, 88
AnyModifier, 88, 89
applicationShellWidgetClass, 38, 40
applicationShellClassRec, 65
application context, 25
ApplicationShell, 58
ApplicationShellPart, 10, 184
ApplicationShellWidget, 61, 63
applicationShellWidgetClass, 38, 39
ApplicationShellWidgetClass, 61
applicationShellWidgetClass, 61
Arg, 34
ArgList, 10, 34, 35, 36, 41, 43

B

Below, 77
Boolean, 9, 127, 182
BottomIf, 77
ButtonPress, 73, 86, 89, 95, 139, 147, 164, 188, 190, 191, 193
ButtonPressMask, 73
ButtonRelease, 86, 95, 164, 188, 190, 191, 193
ConstraintWidget, 8, 9
ConstraintWidgetClass, 8
constraintWidgetClass, 8, 37, 45, 48, 49, 57, 111, 128, 130
CopyFromParent, 45, 65
Core, 1, 2, 3
coreClassRec, 5
CorePart, 2, 4, 10, 69, 174, 175, 178, 184
CoreRec, 10
CoreWidget, 4
Resources, 5
CoreWidgetClass, 3
coreWidgetClass, 37, 45, 148
coreWidgetClassRec, 180
CreateNotify, 190, 193
create_popup_child_proc, 71
CurrentTime, 157, 158, 162, 163
CWBorderWidth, 77
CWHHeight, 77, 78
CWSibling, 77
CWStackMode, 77, 82
CWWidth, 77, 78
CWX, 77
CWY, 77

D

delete_child procedure, 53
Destroy Callbacks, 48, 104
destroy procedure, 49
destroyCallback, 127
DestroyNotify, 190, 193
Dimension, 9
DirectColor, 117
Display, 25, 31
display_accelerator procedure, 142
display_accelerator, 192

E

EastGravity, 45
EnterNotify, 86, 95, 164, 188, 190, 192, 193, 194
EnterWindow, 73
Events, 92
exit, 50
expose procedure, 98
Expose, 97, 98, 132, 164, 190, 193

F

False, 5, 9, 20, 37, 43, 51, 52, 55, 56, 65, 66, 67, 68, 72, 74, 87, 91, 92, 93, 94, 95, 96, 99, 100, 116, 118, 125, 127, 155, 158, 160, 163, 170, 175
FocusChange, 91, 92
FocusIn, 91, 92, 95, 190, 192, 193
FocusOut, 92, 95, 190, 192, 193
font, 5
free, 150

G

Geometry Management, 75
generate_values_hook procedure, 57, 128
Grabbing Input, 86
GrabModeAsync, 73
GrabNotViewable, 89, 90
GraphicsExpose, 193
GraphicsExpose, 97, 98, 100, 101, 102, 103, 164, 190
GravityNotify, 190, 193
GrayScale, 117

I

IconicState, 68, 117
Inheritance, 18, 41, 42, 45, 112
Initialization, 19, 41, 42
initialize procedure, 41, 42
initialize_hook procedure, 43
Input Grabbing, 86
InputOnly, 45
InputOutput, 45
insert_child procedure, 22, 52, 70, 195
Intrinsic.h, 3, 4, 6, 7, 8, 9, 93, 174, 175, 177, 178
IntrinsicP.h, 3, 6, 8, 174, 175

K

key modifier, 190
KeymapNotify, 190, 193
KeyPress, 73, 86, 88, 95, 145, 146, 147, 164, 188, 190, 191, 192, 193
KeyRelease, 86, 95, 145, 146, 164, 188, 190, 192, 193

L

language procedure, 28
LC_ALL, 28, 29
LeaveNotify, 86, 95, 164, 188, 190, 192, 193
libXLa, 1

M

malloc, 150
MapNotify, 190, 193
MappingNotify, 100, 101, 102, 103, 143, 190,
X ToolKit Intrinsics

MapRequest, 190, 193
MenuPopdown, 74
MenuPopup, 73
MotionNotify, 86, 95, 164, 188, 190, 191, 192, 193
multiClickTime, 139

N

NoExpose, 98, 99, 100, 101, 102, 103, 190, 193
None, 6, 67, 68, 91, 147, 156
NormalState, 117
NorthWestGravity, 44, 99
NoSymbol, 144, 146

O

Object, 1, 173, 174, 175
objectClass, 20, 111, 112
ObjectClass, 174
objectClass, 174, 175, 176
ObjectClass, 180
ObjectClassPart, 49, 108, 173, 174, 176
objectClassRec, 5
ObjectClassRec, 174
objectClassRec, 175
ObjectPart, 10, 173, 174, 175, 178, 184
ObjectRec, 10, 175
Opposite, 77
OverrideShell, 58
OverrideShellWidget, 61
OverrideShellWidgetClass, 61
overrideShellWidgetClass, 61

P

ParentRelative, 6
Pixel, 119
pop-up, 69
  child, 69, 70
  list, 69
  shell, 70
Position, 9
PPosition, 66
printf, 166
PropertyNotify, 164, 190, 192, 193
PseudoColor, 117
PSize, 66

Q

query_geometry procedure, 82
R

realize procedure, 45
realloc, 150
Rectangle, 176
RectObj, 176, 177
rectObjClass, 112, 132, 176
RectObjClass, 177
rectObjClass, 177, 178
RectObjClassPart, 49, 108, 176, 177
rectObjClassRec, 5
RectObjClassRec, 177
rectObjClassRec, 178, 180
RectObjPart, 10, 178, 184
RectObjRec, 10, 178
ReparentNotify, 190, 193
resize procedure, 83
ResizeRequest, 190, 193
Resource Management, 108
Resources:
  multiClickTime, 139
  reverseVideo, 116
  selectionTimeout, 33
  synchronous, 33
  xnlLanguage, 29
  xtDefaultFont, 116
  xtDefaultFontSet, 116
  root_geometry_manager procedure, 61
S

Screen, 119
screenConvertArg, 124
SelectionClear, 100, 101, 102, 103, 163, 164, 190, 192, 193
SelectionNotify, 100, 101, 102, 103, 156, 190, 192, 193
SelectionRequest, 100, 101, 102, 103, 155, 160, 190, 192, 193
Selections:
  atomic, 154
  incremental, 159
  MULTIPLE, 155
  TIMESTAMP, 155
  selectionTimeout, 33, 154
  setlocale, 28, 29
  set_values procedure, 131, 133
  set_values_almost procedure, 132
  set_values_hook procedure, 130, 134
  Shell.h, 61
Shell, 58
  create_popup_child_proc, 71
  root_geometry_manager, 61
wm_timeout, 61  
ShellClassExtension, 59, 61  
ShellClassExtensionRec, 59  
ShellClassPart, 23, 61  
shellClassRec, 64  
ShellP.h, 61  
ShellPart, 10, 61, 184  
ShellWidget, 61, 63  
Resources, 64  
ShellWidgetClass, 61  
shellWidgetClass, 61, 71, 73  
sizeof, 23, 110  
special, 5  
StaticColor, 117  
StaticGray, 117  
String Constants:  
  miscellaneous, 211  
  representation types, 210  
  resource classes, 208  
  resource names, 207  
String, 11, 36, 41, 109, 118  
string, 119  
String, 188  
StringDefs.h, 207  
Subclass Chaining, 18  
Substitution, 170  
SubstructureNotify, 47  
Superclass Chaining, 18, 41, 42, 112  
superclass, 14  
synchronous, 33  
T  
TARGETS, 155  
this, 5  
TopIf, 77  
TopLevel, 68  
TopLevelShell, 58  
  resources, 65  
topLevelShellClassRec, 65  
TopLevelShellPart, 10, 182, 183, 184  
TopLevelShellWidget, 61  
TopLevelShellWidgetClass, 61  
topLevelShellWidgetClass, 61, 197  
transientShellClassRec, 65  
TransientShell, 58  
  resources, 65  
TransientShellPart, 182, 183  
TransientShellWidget, 61  
TransientShellWidgetClass, 61  
TransientShellWidgetClass, 61  
Translation tables, 138, 188  
True, 5, 9, 17, 26, 33, 43, 44, 48, 51, 52, 53, 55, 56, 65, 66, 67, 68, 72, 73, 76, 87, 88, 89, 91, 93, 94, 95, 96, 97, 99, 100, 101, 103, 116, 118, 125, 127, 130, 132, 155, 158, 160, 163, 170, 171, 176, 179, 184  
TrueColor, 117  
U  
UnmapNotify, 74, 190, 193  
unrealizeCallback, 47  
USPosition, 66  
USSize, 66  
V  
varargs, 35  
VendorShell, 58  
VendorShellWidget, 61  
VendorShellWidgetClass, 61  
vendorShellWidgetClass, 61  
version, 14  
Visibility, 99  
VisibilityFullyObscured, 99  
VisibilityNotify, 99, 190, 193  
VisibilityPartiallyObscured, 99  
VisibilityUnobscured, 99  
W  
WestGravity, 45  
Widget, 4  
  class extension records, 22  
  class initialization, 20, 113  
WidgetClass, 3, 4  
  widgetClass, 4  
  widgetClassRec, 14  
WidgetList, 54  
WidgetRec, 10  
  widget_class, 10  
  widget_size, 14  
WMShell, 58  
  resources, 64  
  wmShellClassRec, 64  
WMShellPart,, 182  
WMShellPart, 10, 183, 184  
WMShellWidget, 61  
WMShellWidgetClass, 61  
wmShellWidgetClass, 61  
WM_CLASS, 38  
WM_COLORMAP_WINDOWS, 169  
WM_COMMAND, 38  
wm_timeout, 61  
X  
X11/Intrinsic.h, 1, 123  
X11/IntrinsicP.h, 1, 17
X Toolkit Intrinsics

X11/keysymdef.h, 192
X11/Shell.h, 1
X11/StringDefs.h, 1, 10, 11, 108, 110
X11/X.h, 77
X11/Xatom.h, 1
X11/Xaw/Label.h, 1
X11/Xaw/Scrollbar.h, 1
X11/Xresource.h, 118
X11/Xutil.h, 164
XAPPLRESDIR, 30
XA_PRIMARY, 154, 157, 158
XA_SECONDARY, 154
XA_STRING, 154, 156
XClearArea, 130, 132
XCloseDisplay, 27
XConfigureWindow, 47, 54, 76, 77, 80, 81
XCreateFontSet, 116
XCreateGC, 153
XCreateWindow, 44, 45
XDestroyWindow, 47, 48
XFILESEARCHPATH, 171, 185
XFilterEvent, 88, 89, 94
XFreeGC, 49
XFreePixmap, 49
XGrabButton, 89
XGrabKey, 88
XGrabKeyboard, 88, 89
XGrabPointer, 90
XListFonts, 116
XMapRaised, 72
XMatchVisualInfo, 117
xmh, 33
XMoveWindow, 54, 80
XNextEvent, 92
xnILanguage, 29, 171, 186
XOpenDisplay, 27, 32
XParseGeometry, 109
XPeekEvent, 92
XPending, 92
XPointer, 184
XResourceManagerString, 29
XrmGetDatabase, 31, 172
XrmGetDescRec, 32, 184
XrmGetLineResource, 32
XrmGetSetDatabase, 26, 30
XrmGetStoreQuark, 22, 123
XrmValue, 110, 118, 184, 198
XScreenResourceString, 30
XSelectInput, 100, 101, 102, 103
XSetInputFocus, 91, 92
XSetLocaleModifiers, 28
XSetWindowAttributes, 43, 44, 45, 103
XSizeHints, 183
XStdICCTextStyle, 67, 68
XSupportsLocale, 28
XSyncrhonize, 26, 33
XtAcceptFocusProc, 92
XtActionHookId, 138
XtActionHookProc, 137
XtActionList, 135
XtActionProc, 135
XtActionsRec, 135
XtAddActions, 137, 195, 200
XtAddCallback, 49, 105, 175
XtAddCallbacks, 105, 175
XtAddConverter, 126, 198, 199
XtAddEventHandler, 93, 100, 101, 102, 103, 180
XtAddExposeToRegion, 164
XtAddGrab, 72, 86, 87, 94, 180
XtAddInput, 195, 196, 197
XtAddRawEventHandler, 101, 102, 103, 180
XtAddress, 123
XtAddressMode, 123
XtAddTimeOut, 195, 197
XtAddWorkProc, 195, 197
XtAllEvents, 101, 103
XtAllocateGC, 151, 152, 153
XtAlmostProc, 132
XtAppDelegateInput, 85
XtAppDelegateTimeOut, 86
XtAppDelegateActionHook, 137
XtAppDelegateActionHook, 137, 138
XtAppDelegateActions, 136, 137, 200
XtAppDelegateConverter, 126, 198, 199
XtAppDelegateInput, 84, 85, 93, 197
XtAppDelegateTimeOut, 49, 85, 86, 197
XtAppDelegateWorkProc, 96, 197
XtAppDelegateContext, 25
XtAppDelegateCreateShell, 24, 38, 39, 40, 112, 180, 181, 196, 197
XtAppDelegateError, 169, 203
XtAppDelegateErrorMsg, 121, 167, 169, 202, 203
XtAppDelegateErrorDatabase, 165, 201
XtAppDelegateErrorDatabaseText, 165, 166, 201
XtAppDelegateGetSelectionTimeout, 153, 154, 201
XtAppDelegateInitialize, 25, 39, 40, 41, 181, 186, 196
XtAppDelegateMainLoop, 84, 93, 94, 196
XtAppDelegateNextEvent, 86, 93, 94, 96, 196
XtAppDelegatePeekEvent, 92, 93, 196
XtAppDelegatePending, 92, 196
XtAppDelegateProcessEvent, 86, 92, 93, 96, 196
XtAppDelegateReleaseCacheRefs, 125
XtAppDelegateSetErrorHandler, 168, 203
XtAppDelegateSetErrorMsgHandler, 167, 202
XtAppDelegateSetFallbackResources, 31, 32, 40
XtAppDelegateSetSelectionTimeout, 153, 201
XtAppDelegateSetTypeConverter, 122, 123, 199
XtAppDelegateSetWarningHandler, 169, 203
XtAppDelegateSetWarningMsgHandler, 167, 168, 202
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtAppWarning</td>
<td>169, 203</td>
</tr>
<tr>
<td>XtAppWarningMsg</td>
<td>118, 121, 168, 169, 202, 203</td>
</tr>
<tr>
<td>XtArgsFunc</td>
<td>134</td>
</tr>
<tr>
<td>XtArgsProc</td>
<td>43, 128</td>
</tr>
<tr>
<td>XtArgVal</td>
<td>9, 34, 35, 36</td>
</tr>
<tr>
<td>XtAugmentTranslations</td>
<td>140, 180</td>
</tr>
<tr>
<td>XtBaseOffset</td>
<td>123, 124</td>
</tr>
<tr>
<td>XtBuildEventMask</td>
<td>103, 180</td>
</tr>
<tr>
<td>XtButtonBoxAddButton</td>
<td>195</td>
</tr>
<tr>
<td>XtButtonBoxDeleteButton</td>
<td>195</td>
</tr>
<tr>
<td>XtCacheAll</td>
<td>121, 125, 199</td>
</tr>
<tr>
<td>XtCacheByDisplay</td>
<td>121, 125, 127</td>
</tr>
<tr>
<td>XtCacheNone</td>
<td>121, 125</td>
</tr>
<tr>
<td>XtCacheRef</td>
<td>125, 126, 127</td>
</tr>
<tr>
<td>XtCacheRefCount</td>
<td>122, 125</td>
</tr>
<tr>
<td>XtCacheType</td>
<td>121</td>
</tr>
<tr>
<td>XtCallAcceptFocus</td>
<td>92, 181</td>
</tr>
<tr>
<td>XtCallActionProc</td>
<td>138, 146, 147, 148, 180</td>
</tr>
<tr>
<td>XtCallbackExclusive</td>
<td>71, 72, 73, 181</td>
</tr>
<tr>
<td>XtCallbackHasNone</td>
<td>105, 107</td>
</tr>
<tr>
<td>XtCallbackHasSome</td>
<td>107</td>
</tr>
<tr>
<td>XtCallbackList</td>
<td>104</td>
</tr>
<tr>
<td>XtCallbackNoList</td>
<td>107</td>
</tr>
<tr>
<td>XtCallbackNonexclusive</td>
<td>71, 72, 73, 181</td>
</tr>
<tr>
<td>XtCallbackPopdown</td>
<td>73, 74, 181</td>
</tr>
<tr>
<td>XtCallbackProc</td>
<td>49, 104</td>
</tr>
<tr>
<td>XtCallbackReleaseCacheRef</td>
<td>126, 127</td>
</tr>
<tr>
<td>XtCallbackReleaseCacheRefList</td>
<td>126</td>
</tr>
<tr>
<td>XtCallbackStatus</td>
<td>105</td>
</tr>
<tr>
<td>XtCallbackListList</td>
<td>105, 107, 175</td>
</tr>
<tr>
<td>XtCallbackList</td>
<td>105, 107, 175</td>
</tr>
<tr>
<td>XtCallback</td>
<td>123, 124, 125, 126, 200</td>
</tr>
<tr>
<td>XtAlloc</td>
<td>49, 150, 151, 158, 163</td>
</tr>
<tr>
<td>XtCancelConvertSelectionProc</td>
<td>161</td>
</tr>
<tr>
<td>XtCaseProc</td>
<td>143, 144, 145, 184</td>
</tr>
<tr>
<td>XtCheckSubclass</td>
<td>2, 17, 18, 71, 73, 175</td>
</tr>
<tr>
<td>XtClass</td>
<td>16, 17, 175, 176</td>
</tr>
<tr>
<td>XtCloseDisplay</td>
<td>27, 121, 127</td>
</tr>
<tr>
<td>XtCompositeExtensionVersion</td>
<td>7</td>
</tr>
<tr>
<td>XtConfigureProc</td>
<td>48, 87, 88, 89, 91, 93, 94, 164, 179, 181</td>
</tr>
<tr>
<td>XtCreateApplicationContext</td>
<td>24, 25, 137</td>
</tr>
<tr>
<td>XtCreateApplicationShell</td>
<td>195, 197</td>
</tr>
<tr>
<td>XtCreateManagedWidget</td>
<td>51, 54, 55, 52, 122, 178, 179, 181</td>
</tr>
<tr>
<td>XtCreatePopupChildProc</td>
<td>71</td>
</tr>
<tr>
<td>XtCreatePopupShell</td>
<td>39, 70, 180, 181</td>
</tr>
<tr>
<td>XtCreateWidget</td>
<td>5, 20, 34, 36, 37, 38, 44, 51, 52, 54, 57, 104, 108, 112, 113, 122, 127, 175, 176, 179, 195</td>
</tr>
<tr>
<td>XtCreateWindow</td>
<td>45, 47, 180</td>
</tr>
<tr>
<td>XtCvtColorToPixel</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtUintToBool</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtUintToBoolean</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtUintToColor</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtUintToDimension</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtUintToFloat</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtIntToFont</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtIntToPixel</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtIntToPixmap</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtIntToPosition</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtIntToShort</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtIntToUnsignedChar</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtColorToColor</td>
<td>117</td>
</tr>
<tr>
<td>XtCvtStringToAcceleratorTable</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToAtom</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToBoolean</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToCursor</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToDimension</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToDisplay</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToFile</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToFloat</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToFont</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToFontSet</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToFontStruct</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToFontStruct</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToInitialState</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToInt</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToPixel</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToPosition</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToPosition</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToShort</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToTranslationTable</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToUnsignedChar</td>
<td>115</td>
</tr>
<tr>
<td>XtCvtStringToVisual</td>
<td>115</td>
</tr>
<tr>
<td>XtCtQueryOnly</td>
<td>76, 77, 78, 79, 80</td>
</tr>
<tr>
<td>XtDatabase</td>
<td>31</td>
</tr>
<tr>
<td>XtDefaultBackground</td>
<td>5, 33, 116, 120</td>
</tr>
<tr>
<td>XtDefaultFont</td>
<td>116, 120</td>
</tr>
<tr>
<td>XtDefaultFontSet</td>
<td>116</td>
</tr>
<tr>
<td>XtDefaultForeground</td>
<td>5, 33, 110, 116, 120</td>
</tr>
<tr>
<td>XtDestroyWidget</td>
<td>44</td>
</tr>
<tr>
<td>XtDestroyApplicationContext</td>
<td>25, 27, 50</td>
</tr>
<tr>
<td>XtDestroyGC</td>
<td>200</td>
</tr>
<tr>
<td>XtDestroyWidget</td>
<td>24, 48, 49, 50, 51, 53, 55, 57, 69, 176</td>
</tr>
<tr>
<td>XtDestructor</td>
<td>120</td>
</tr>
<tr>
<td>XtDirectConvert</td>
<td>123, 126, 199, 200</td>
</tr>
<tr>
<td>XtDisownSelection</td>
<td>158, 159, 163, 180</td>
</tr>
<tr>
<td>XtDispatchEvent</td>
<td>48, 87, 88, 89, 91, 93, 94, 164, 217</td>
</tr>
</tbody>
</table>

**X Toolkit Intrinsics**

**X Toolkit Intrinsics**
X Toolkit Intrinsics

X11 Release 5

185, 196
XtDisplay, 46, 180
XtDisplayInitialize, 24, 25, 26, 27, 28, 29, 31, 32, 33, 38, 39, 40, 139, 154, 166, 167, 171, 186
XtDisplayOfString Object, 46, 176
XtDisplayStringConversionWarning, 121, 198
XtDisplayToApplicationContext, 118, 121
XtEnum, 9, 182
XtError, 167, 202, 203
XtErrorHandler, 168
XtErrorMSG, 18, 150, 151, 202
XtErrorMsg, 165
XtDisplayStringConversionWarning, 121, 198
XtDisplayToApplicationContext, 118, 121
XtGeometryMask, 76
XtGeometryResult, 77
XtGeometryYes, 61, 76, 77, 79, 80, 82, 130, 180
XtGetActionKeysym, 145, 146
XtGetApplicationNameAndClass, 165, 167, 172
XtGetApplicationResources, 114, 115, 119, 122, 124, 176
XtGetConstraintResourceList, 111, 175
XtGetExceptionDatabase, 201
XtGetExceptionDatabaseText, 201
XtGetGC, 143, 144, 146
XtGetMultiClickTime, 139
XtGetResourceList, 111, 175
XtGetSelectionRequest, 155, 160, 180
XtGetSelectionTimeout, 195, 201
XtGetSelection Value, 156, 157, 162, 180
XtGetSelectionValueIncremental, 161, 162, 180
XtGetSelectionValues, 156, 157, 180
XtGetSubclass, 156, 157, 180
XtGetSubresources, 113, 114, 119, 122, 124, 176
XtGetSubvalues, 129
XtGrabButton, 89, 90, 94, 147, 180
XtGrabExclusive, 72, 73, 74
XtGrabKey, 87, 88, 91, 94, 147, 180
XtGrabKeyboard, 88, 89, 91, 180
XtGrabKind, 71
XtGrabNone, 68, 73
XtGrabNonexclusive, 72, 73, 74
XtGrabPointer, 90, 180
XtHasCallbacks, 107, 175
XtMAll, 93
XtMALternateInput, 92, 93
XtImmediate, 123
XtMTimer, 92, 93
XtMXEvent, 92, 93
XtInherit, 20
XtInheritAcceptFocus, 21
XtInheritChangeManaged, 21
XtInheritDeleteChild, 21
XtInheritDisplayAccelerator, 21
XtInheritExpose, 21
XtInheritGeometryManager, 21
XtInheritInsertChild, 21
XtInheritQueryGeometry, 21
XtInheritRealize, 21
XtInheritResize, 21
XtInheritRootGeometryManager, 21, 61
XtInheritSetValuesAlmost, 21, 132
XtInheritTranslations, 21, 138
XtInitialize, 186, 195, 196, 197, 200, 201
XtInitializeWidgetClass, 20, 175
XtInputProc, 41, 42, 182
XtInputCallbackProc, 85
XtInputExceptMask, 85
XtInputReadMask, 85
XtInputWriteMask, 85
XtInsertEventHandler, 101, 103, 180
XtInsertRawEventHandler, 101, 102, 103, 180
XtInstallAccelerators, 142, 180
XtInstallAllAccelerators, 142, 143, 180
XtIsApplicationShell, 17, 175
XtIsComposite, 17, 175
XtIsConstraint, 17, 175
XtIsManaged, 55, 175
XtIsObject, 17, 175
XtIsOverrideShell, 17, 175
XtIsRealized, 44, 175
XtIsRectObj, 17, 175
XtIsSensitive, 95, 175
XtIsShell, 17, 175
XtIsSubclass, 17, 175
XtIsTopLevelShell, 17
XtIsTopLevelShell, 17, 175
XtIsTransientShell, 17, 175
XtIsVendorShell, 17, 175
XtIsWidget, 17, 175
X Toolkit Intrinsics

XtDisplay, 24, 25, 27, 32, 40, 186, 196
XtWindow, 8, 125
XtPointer, 9, 110, 126, 182
XtPopdown, 66, 73, 74, 181
XtPopdownID, 74
XtPopup, 66, 68, 71, 72, 73, 86, 181
XtPopupSpringLoaded, 71, 72, 73, 181
XtProc, 19
XtProcedureArg, 123
XtProcessEvent, 195, 196
XtQueryGeometry, 81, 82, 33, 179, 183
XtQueryOnly, 180
XtRAcceleratorTable, 109, 115
XtTranslateTable, 180, 195, 196
XtAtom, 109, 115
XtBimap, 109
XtBool, 109, 115, 117
XtBoolean, 109, 115, 117
XtCallback, 105, 107, 109
XtCallback, 109
XtColor, 109, 117
XtColorMap, 109
XtCursor, 109, 115
XtDimension, 109, 115, 117
XtDisplay, 109, 115
XtRealizeProc, 44
XtRealizeWidget, 24, 34, 43, 44, 45, 53, 71, 72, 79, 99, 103, 181
XtRealloc, 150, 151, 158, 163
XtEditMode, 110
XtRegisterCaseConverter, 145
XtRegisterGrabAction, 73, 146, 147
XtReleaseGC, 49, 153, 176, 200
XtRemoveActionHook, 138
XtRemoveAllCallbacks, 106, 175
XtRemoveCallback, 49, 106, 175
XtRemoveCallbacks, 106, 175
XtRemoveEventHandler, 49, 101, 101, 180
XtMoveWindow, 81, 181
XtRGeometry, 109, 115
XtRInt, 109, 115
XtRJustify, 110

219
<table>
<thead>
<tr>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtRLongBoolean, 109</td>
</tr>
<tr>
<td>XtRObject, 109</td>
</tr>
<tr>
<td>XtROrientation, 110</td>
</tr>
<tr>
<td>XtRPixel, 109, 115, 117</td>
</tr>
<tr>
<td>XtRPixmap, 109, 117</td>
</tr>
<tr>
<td>XtRPointer, 109</td>
</tr>
<tr>
<td>XtRPosition, 109, 115, 117</td>
</tr>
<tr>
<td>XtRScreen, 109</td>
</tr>
<tr>
<td>XtRShort, 109, 115, 117</td>
</tr>
<tr>
<td>XtRString, 109, 110, 115, 116, 118</td>
</tr>
<tr>
<td>XtRStringArray, 109</td>
</tr>
<tr>
<td>XtRStringTable, 109</td>
</tr>
<tr>
<td>XtRTranslation, 110</td>
</tr>
<tr>
<td>XtRTranslationTable, 109, 115</td>
</tr>
<tr>
<td>XtRUnsignedChar, 109, 115, 117</td>
</tr>
<tr>
<td>XtRVisual, 109, 115, 117</td>
</tr>
<tr>
<td>XtRWidget, 109</td>
</tr>
<tr>
<td>XtRWidgetClass, 109</td>
</tr>
<tr>
<td>XtRWidgetList, 109</td>
</tr>
<tr>
<td>XtRWindow, 109</td>
</tr>
<tr>
<td>XtArg, 2, 34, 35</td>
</tr>
<tr>
<td>XtArgError, 203</td>
</tr>
<tr>
<td>XtArgerrorMsg, 201</td>
</tr>
<tr>
<td>XtArgKeyboardFocus, 91, 176, 181</td>
</tr>
<tr>
<td>XtArgKeyTranslate, 144</td>
</tr>
<tr>
<td>XtArgLanguage, 24, 25, 28, 29, 185</td>
</tr>
<tr>
<td>XtArgMappedWhenManaged, 44, 51, 56, 181</td>
</tr>
<tr>
<td>XtArgMultiClickTime, 139</td>
</tr>
<tr>
<td>XtArgSelectionTimeout, 195, 200</td>
</tr>
<tr>
<td>XtArgSensitive, 71, 73, 74, 95, 179</td>
</tr>
<tr>
<td>XtArgSubvalues, 133</td>
</tr>
<tr>
<td>XtArgTypeConverter, 199</td>
</tr>
<tr>
<td>XtArgTypeConverter, 122, 123, 199</td>
</tr>
<tr>
<td>XtArgValues, 10, 47, 56, 57, 71, 75, 95, 104, 108, 111, 130, 131, 132, 134, 141, 176, 179</td>
</tr>
<tr>
<td>XtArgValuesFunc, 131, 133, 182</td>
</tr>
<tr>
<td>XtArgWarningHandler, 203</td>
</tr>
<tr>
<td>XtArgWarningMsgHandler, 202</td>
</tr>
<tr>
<td>XtArgWMColorMapWindows, 169, 170, 181</td>
</tr>
<tr>
<td>XtArgExtensionVersion, 61</td>
</tr>
<tr>
<td>XtArgDonChange, 77, 82</td>
</tr>
<tr>
<td>XtArgSpecificationRelease, 14, 182</td>
</tr>
<tr>
<td>XtArgStringConversionWarning, 198</td>
</tr>
<tr>
<td>XtArgProc, 142</td>
</tr>
<tr>
<td>XtArgSuperclass, 17, 22, 175, 176</td>
</tr>
<tr>
<td>XtArgTimerCallbackProc, 86</td>
</tr>
<tr>
<td>XtArgToolkitInitialize, 24, 25, 40, 196</td>
</tr>
<tr>
<td>XtArgTranslateCoords, 164, 179</td>
</tr>
<tr>
<td>XtArgTranslateKey, 144</td>
</tr>
<tr>
<td>XtArgTranslateKeyCode, 144, 146</td>
</tr>
<tr>
<td>XtTranslations, 140</td>
</tr>
<tr>
<td>XtArgTypeConverter, 118, 198</td>
</tr>
<tr>
<td>XtUngrabButton, 89, 90, 180</td>
</tr>
<tr>
<td>XtUngrabKey, 88, 180</td>
</tr>
<tr>
<td>XtUngrabKeyboard, 88, 89, 91, 94, 180</td>
</tr>
<tr>
<td>XtUngrabPointer, 89, 90, 94, 180</td>
</tr>
<tr>
<td>XtUninstallTranslations, 141, 180</td>
</tr>
<tr>
<td>XtUnmanageChildren, 44</td>
</tr>
<tr>
<td>XtUnmanageChild, 48, 51, 55, 179</td>
</tr>
<tr>
<td>XtUnmanageChildren, 51, 55, 179, 195</td>
</tr>
<tr>
<td>XtUnmapWidget, 50, 56, 181</td>
</tr>
<tr>
<td>XtUnrealizeWidget, 44, 47, 181, 183</td>
</tr>
<tr>
<td>XtUnspecifiedPixmap, 5, 6, 44</td>
</tr>
<tr>
<td>XtUnspecifiedShellInt, 66, 67</td>
</tr>
<tr>
<td>XtUnspecifiedWindow, 66, 67</td>
</tr>
<tr>
<td>XtUnspecifiedWindowGroup, 67</td>
</tr>
<tr>
<td>XtVaAppCreateShell, 39, 180, 181</td>
</tr>
<tr>
<td>XtVaAppInitialize, 39, 40, 41, 181, 186</td>
</tr>
<tr>
<td>XtVaCreateArgList, 36</td>
</tr>
<tr>
<td>XtVaCreateManagedWidget, 54, 55, 178, 179, 181</td>
</tr>
<tr>
<td>XtVaCreatePopupShell, 70, 180, 181</td>
</tr>
<tr>
<td>XtVaCreateWidget, 37, 38, 44, 175, 176</td>
</tr>
<tr>
<td>XtVaGetApplicationResources, 115, 124, 176</td>
</tr>
<tr>
<td>XtVaGetSubresources, 114, 124, 176</td>
</tr>
<tr>
<td>XtVaGetSubvalues, 129</td>
</tr>
<tr>
<td>XtVaGetValues, 128, 129, 176</td>
</tr>
<tr>
<td>XtVaNestedList, 36</td>
</tr>
<tr>
<td>XtVaArgList, 36</td>
</tr>
<tr>
<td>XtVaSetArgList, 133</td>
</tr>
<tr>
<td>XtVaSetValues, 130, 131, 134, 176</td>
</tr>
<tr>
<td>XtVaTypedArg, 36, 41, 43, 128, 129, 133</td>
</tr>
<tr>
<td>XtVersion, 14</td>
</tr>
<tr>
<td>XtVersionDonCheck, 14</td>
</tr>
<tr>
<td>XtWarning, 28, 168, 202, 203</td>
</tr>
<tr>
<td>XtWarningMsg, 198, 202</td>
</tr>
<tr>
<td>XtWidgetBaseOffset, 123, 124</td>
</tr>
<tr>
<td>XtWidgetClassProc, 19</td>
</tr>
<tr>
<td>XtWidgetGeometry, 76, 77, 78, 82</td>
</tr>
<tr>
<td>XtWidgetProc, 49, 52, 53, 83</td>
</tr>
<tr>
<td>XtWidgetToApplicationContext, 25, 176</td>
</tr>
<tr>
<td>XtWindow, 46, 180</td>
</tr>
<tr>
<td>XtWindowToObject, 46, 176</td>
</tr>
<tr>
<td>XtWindowToWidget, 165, 181</td>
</tr>
<tr>
<td>XtWorkProc, 96</td>
</tr>
<tr>
<td>XT_CONVERT_FAIL, 156, 162</td>
</tr>
<tr>
<td>XtUngrabButton, 90</td>
</tr>
<tr>
<td>XtUngrabKey, 88</td>
</tr>
<tr>
<td>XtUngrabKeyboard, 89</td>
</tr>
<tr>
<td>XtUngrabPointer, 90</td>
</tr>
<tr>
<td>XUSERFILESEARCHPATH, 30, 185</td>
</tr>
</tbody>
</table>
XWMGeometry, 66, 67

FALSE, 9
LANG, 30
NULLQUARK, 6, 8, 22, 37, 57, 61, 128
RESOURCE_NAME, 27
TRUE, 9
WM_CHANGE_STATE, 68
WM_COMMAND, 68
WM_ICON_NAME, 68
WM_NAME, 67
WM_TRANSIENT_FOR, 58, 67, 68
XAPPLRESDIR, 30, 31
XENVIRONMENT, 30
XFILESEARCHPATH, 171
XUSERFILESEARCHPATH, 30, 31

_XtDefaultError, 168
_XtDefaultErrorMsg, 167
_XtDefaultWarning, 169
_XtDefaultWarningMsg, 168
_XtError, 203
_XtInherit, 20
_XtWarning, 203
Bitmap Distribution Format

Version 2.1

MIT X Consortium Standard

X Version 11, Release 5


Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

The Bitmap Distribution Format (BDF), Version 2.1, is an X Consortium standard for font interchange, intended to be easily understood by both humans and computers.

File Format

Character bitmap information will be distributed in an USASCII-encoded, human-readable form. Each file is encoded in the printable characters (octal 40 through 176) of USASCII plus carriage return and linefeed. Each file consists of a sequence of variable-length lines. Each line is terminated either by a carriage return (octal 015) and linefeed (octal 012) or by just a linefeed.

The information about a particular family and face at one size and orientation will be contained in one file. The file begins with information pertaining to the face as a whole, followed by the information and bitmaps for the individual characters.

A font bitmap description file has the following general form, where each item is contained on a separate line of text in the file. Tokens on a line are separated by spaces. Keywords are in upper-case, and must appear in upper-case in the file.

1. The word STARTFONT followed by a version number indicating the exact file format used. The version described here is 2.1.
2. Lines beginning with the word COMMENT may appear anywhere between the STARTFONT line and the ENDFONT line. These lines are ignored by font compilers.
3. The word FONT followed by either the XLFD font name (as specified in part III) or some private font name. Creators of private font name syntaxes are encouraged to register unique font name prefixes with the X Consortium to prevent naming conflicts. Note that the name continues all the way to the end of the line and may contain spaces.
4. The word SIZE followed by the point size of the characters, the x resolution, and the y resolution of the device for which these characters were intended.
5. The word FONTBOUNDINGBOX followed by the width in x, height in y, and the x and y displacement of the lower corner from the origin. (See the examples in the next section.)
6. Optionally, the word STARTPROPERTIES followed by the number of properties (p) that follow.
7. Then come p lines consisting of a word for the property name followed by either an integer or string surrounded by double-quote (octal 042). Internal double-quote characters are indicated by using two in a row.
Properties named FONT_ASCENT, FONT_DESCENT, and DEFAULT_CHAR should be pro-
vided to define the logical font-ascent and font-descent and the default-char for the font. These
properties will be removed from the actual font properties in the binary form produced by a com-
piler. If these properties are not provided, a compiler may reject the font or may compute (arbi-
trary) values for these properties.

8. The property section, if it exists, is terminated by ENDPREPROPERTIES.

9. The word CHARS followed by the number of character segments (c) that follow.

10. Then come c character segments of the form:
   a. The word STARTCHAR followed by up to 14 characters (no blanks) of descriptive name of
the glyph.
   b. The word ENCODING followed by one of the following forms:
      i. <n> – the glyph index, that is, a positive integer representing the character code used
to access the glyph in X requests, as defined by the encoded character set given by
the CHARSET_REGISTRY-CHARSET_ENCODING font properties for XLFD con-
forming fonts. If these XLFD font properties are not defined, the encoding scheme is
font-dependent.
      ii. -1 <n> – equivalent to form above. This syntax is provided for backward compatibil-
ity with previous versions of this specification and is not recommended for use with
new fonts.
      iii. -1 – an unencoded glyph. Some font compilers may discard unencoded glyphs, but,
in general, the glyph names may be used by font compilers and X servers to imple-
ment dynamic mapping of glyph repertoires to character encodings as seen through
the X protocol.
   c. The word SWIDTH followed by the scalable width in x and y of character. Scalable
widths are in units of 1/1000th of the size of the character. If the size of the character is p
points, the width information must be scaled by p/1000 to get the width of the character in
printer’s points. This width information should be considered as a vector indicating the
position of the next character’s origin relative to the origin of this character. To convert
the scalable width to the width in device pixels, multiply SWIDTH times p/1000 times r/72,
where r is the device resolution in pixels per inch. The result is a real number giving the
ideal print width in device pixels. The actual device width must of course be an integral
number of device pixels and is given in the next entry. The SWIDTH y value should
always be zero for a standard X font.
   d. The word DWIDTH followed by the width in x and y of the character in device units.
Like the SWIDTH, this width information is a vector indicating the position of the next
character’s origin relative to the origin of this character. Note that the DWIDTH of a given
“hand-tuned” WYSIWYG glyph may deviate slightly from its ideal device-independent
width given by SWIDTH in order to improve its typographic characteristics on a display.
The DWIDTH y value should always be zero for a standard X font.
   e. The word BBX followed by the width in x (BBw), height in y (BBh) and x and y displace-
ment (BBox, BBoy) of the lower left corner from the origin of the character.
   f. The optional word ATTRIBUTES followed by the attributes as 4 hex-encoded characters.
The interpretation of these attributes is undefined in this document.
   g. The word BITMAP.
   h. h lines of hex-encoded bitmap, padded on the right with zeros to the nearest byte (that is,
multiple of 8).
   i. The word ENDCCHAR.

11. The file is terminated with the word ENDFONT.
Metric Information

Figures 1 and 2 best illustrate the bitmap format and character metric information.
Figure 1: An example of a descender
Figure 2: An example with the origin outside the bounding box

BBh = 6, BBw = 4, BBox = +2, BBoy = +12
DWIDTH = 5 0
SWIDTH = 223 0
An Example File

The following is an abbreviated example of a bitmap file containing the specification of two characters (the j and quoterright in figures 1 and 2).

```
STARTFONT 2.1
COMMENT This is a sample font in 2.1 format.
FONT -Adobe-Helvetica-Bold-R-Normal--24-240-75-75-P-65-ISO8859-1
SIZE 24 75 75
FONTBOUNDINGBOX 9 24 -2 -6
STARTPROPERTIES 19
FOUNDRY "Adobe"
FAMILY "Helvetica"
WEIGHT_NAME "Bold"
SLANT "R"
SETWIDTH_NAME "Normal"
ADD_STYLE_NAME ""
PIXEL_SIZE 24
POINT_SIZE 240
RESOLUTION_X 75
RESOLUTION_Y 75
SPACING "P"
AVERAGE_WIDTH 65
CHARSET_REGISTRY "ISO8859"
CHARSET_ENCODING "1"
MIN_SPACE 4
FONT_ASCENT 21
FONT_DESCENT 7
COPYRIGHT "Copyright (c) 1987 Adobe Systems, Inc."
NOTICE "Helvetica is a registered trademark of Linotype Inc."
ENDPROPERTIES
CHARS 2
STARTCHAR j
ENCODING 106
SWIDTH 355 0
DWIDTH 8 0
BBX 9 22 -2 -6
BITMAP
0380
0380
0380
0380
0000
0700
0700
0700
0700
0E00
0E00
0E00
0E00
0E00
1C00
1C00
1C00
```