Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

XWIP REFERENCE MANUAL
VERSION 0.4, THE OGIVE EDITION

Ted Kim

October 1988
CSD-880079
(Rev. 4/89)

XWIP
Reference Manual
Version 0.4
The Ogive Edition

Ted Kim

April 1, 1989

This software work was developed at UCLA with support in part from
DARPA Contract F29601-87-C-0072.

Abstract

This manual describes an interface to the X Window System [1] for Prolog. The X Window System is a network-based window system providing a desktop style of user interface and graphics. XWIP provides a low level interface to X for Prolog similar to that provided by "Xlib" for the C language. XWIP is designed for use with version 11 of the X Window System. Higher level interfaces (such as *toolkits*) are built on top of this one and are outside the scope of this document.

XWIP is implemented in the C language using the C language foreign function interface from Quintus Prolog.[2] Almost any Prolog which supports the Quintus style interface can use this package with few restrictions. In particular, SICStus Prolog was used in the development of this system.[3] This document is a reference manual. As such, it is not a tutorial or user's guide to X or Prolog.

[1] Scheifler, R.W. and Gettys, J., "The X Window System", *ACM Transactions on Graphics*, vol. 5, no. 2, pp. 79-109, April 1986.

X Window System is a trademark of the Massachusetts Institute of Technology.

[2] *Quintus* and *Quintus Prolog* are trademarks of Quintus Computer Systems, Inc.

[3] Carlsson, M. and Widen, J., "SICStus Prolog User's Manual", Swedish Institute of Computer Science, Research Report SICS R88007.

Contents

1 Introduction 1

2 Using XWIP 1

3 Implementation 2

4 Conventions 3

5 Data Types 4

6 Connections 7

7 Windows 11

8 Atoms and Properties 14

9 Pixmaps 16

10 Color 17

11 Graphics Contexts 19

12 Graphics 21

13 Images 23

14 Text 24

15 Cursors 27

16 The Pointer 28

17 The Keyboard 29

18 Window Manager Support 31

19 Events 34

20 Miscellaneous 40

A Xlib Equivalents 44

B X Font Cursors 46

1 Introduction

This manual describes the "X Window Interface for Prolog" system, hereafter referred to by its acronym, "XWIP" (pronounced "X-whip"). XWIP was formerly called PX. The name was changed, when it was found that another system also used that name.

The system provides an interface to the X Window System for Prolog.[4] The X Window System is a network-based window system providing a desktop style of user interface and graphics. XWIP provides a low level interface to X for Prolog similar to that provided by "Xlib" for the C language. XWIP is designed for use with version 11 of the X Window System. Higher level interfaces (such as *toolkits*) are built on top of this one and are outside the scope of this document.

XWIP is implemented in the C language using the C language foreign function interface from Quintus Prolog. Almost any Prolog which supports the Quintus style interface can use this package with few restrictions. In particular, SICStus Prolog was used in the development of this system.

This document is a reference manual. XWIP descriptions in this manual assume the user is familiar with X and programming with X in the C language and some variety of Prolog. This manual is not a tutorial or user's guide to X or Prolog. The XWIP programmer should have access to to X documentation, especially the Xlib manual.[5]

2 Using XWIP

Generally, XWIP is preloaded into Prolog. The preloaded SICStus version is called "spx" and the Quintus version is called "qpx". These preloaded versions are found in the same access path used for X binaries. On UNIX[6] systems, the default location for X binaries is the directory "/usr/bin/X11". An alternate method is to consult the "spx" or "qpx" Prolog files from the XWIP source directory. Before actually starting XWIP, UNIX users may want to specify a default connection name by setting their DISPLAY environment variable. Similarly, users may want to specify a resource database by setting their XENVIRONMENT environment variable.

XWIP does not implement the "convenience" functions of Xlib, which are just wrappers around the general protocol functions. For instance, the Xlib function XSetWindowBackground is just a particular request of the XChangeWindowAttributes. Similarly, a XWIP user should use xSetWindow to set the window background. The problem with convenience functions is that programmers tend to

[4]This work was supported by the TANGRAM project, DARPA Contract F29601-87-C-0072.

[5]Gettys, J., Newman, R. and Scheifler, R.W., "Xlib – C Language X Interface, X Version 11, Release 3", MIT.

[6]Unix is a trademark of AT&T.

use them when its not appropriate. Where a general function can batch several requests into one protocol request, convenience functions cannot.

XWIP detects whatever errors it can before making a request to the server. Such errors will cause XWIP predicates to fail. Other errors can be detected only by the server. These errors occur asynchronously and do not cause predicate failures (except *xOk*, whose sole purpose is to detect such errors). Both types cause diagnostic messages which are printed to the Prolog standard error output. Example error messages are given below.

[ERROR xSetWindow/3: no such connection]
: This is an example of a XWIP detected error. *xSetWindow*/3 is the name and arity of the predicate. The rest of the message is an explanation of the error.

[ERROR detected by X Server: not a valid window ID]
[ERROR generated by ChangeWindowAttributes request with serial 5]
: This is an example of a server-detected error. The first line gives an explanation of the error. The second line gives an indication of what caused the error in terms of the actual protocol request and its serial number. Predicates do not necessarily map one to one with protocol requests. The current request serial number can be determined by using the *xQueryConnection* predicate.

3 Implementation

The foreign function interface was used in preference to direct modification of the Prolog interpreter for three reasons. First, it was much simpler to write. Second, it is relatively stable compared to the internal structure of Prolog interpreters, which seem to undergo major changes with each new release. Finally, it enhances portability between versions of Prolog.

On the negative side, this approach is a bit less efficient. Also, the foreign function interface is somewhat deficient in some areas. These deficiencies lead to an imperfect match with some requirements of X.

Since communication in X between the client and server take place through network services, the delay for any particular request may be large. In order to maintain good throughput, many X requests are asynchronous. Errors detected by the server are flagged by XWIP when the error events are received, which may be long after the original request was made. As a result, XWIP predicates may succeed, but not actually perform as requested. Thus, success of XWIP predicates should be taken to mean that the form of a request was proper, but not necessarily its content. XWIP provides facilities to allow the synchronization of requests and the testing of the server error state.

XWIP uses *X identifiers* (XIDs) as defined in the X Protocol Document.[7]

[7]Scheifler, R.W., "X Window System Protocol – X Version 11, Release 3", MIT.

These identifiers are used by the client to reference server maintained objects. XWIP uses Prolog integers to represent these identifiers. However, because of the limited precision of integers in many Prologs, there exists a potential problem with misrepresentation of identifiers. The protocol defines X identifiers as unsigned 32 bit integers with the top 3 bits set to zero. X identifiers are generally not checked for validity by XWIP for efficiency reasons. Therefore, if a Prolog system has significantly less than 29 bits of precision, then it is unsuitable for use with the current implementation of XWIP.[8]

Some other X values are represented by 32 bit unsigned integers. Prolog systems generally provide only signed integers. Using signed integers to represent unsigned values can result in trouble, especially if sign extension occurs. In general, there is no problem as long as very large values are not used. For example, a client would have to use millions of X atoms before precision became an issue. In those few cases, where large values are commonly used, a special method of representation, called a *split* structure is provided below. As a practical matter, XWIP may give incorrect results if the actual precision of Prolog integers is less than 25 bits including sign.

XWIP makes use of a number of client maintained data structures provided by Xlib. In C, these structures would be referred to by pointers. Aside from possible precision problems, pointer values could also be represented as integers. However, since there is no guarantee that valid pointers would be passed to the XWIP routines, this method is dangerous. Instead, integer keys similar to UNIX file descriptors are used to represent these Xlib structures. The validity of such descriptors is checked by XWIP.

4 Conventions

XWIP makes use of some internal predicates, all of which are prefixed by "px". In some versions, no user visible internal predicates exist. However, to be on the safe side, the user should avoid use of predicates with such names. All external predicates and data structures are prefixed with "x".

XWIP was meant to provide an Xlib style interface. Thus, unless otherwise stated, XWIP routines use the exact same arguments as their Xlib counterparts, with some simple translations to the Prolog world. Generally, descriptions of XWIP predicates only describe differences and clarifications from the equivalent Xlib function description. In some cases, the XWIP predicate is named differently from its Xlib counterpart. These cases are listed in Appendix A. Names were changed to make them more consistent or, in a few cases, shorter.

This manual uses the following format for its descriptions:

xDataStructure(Arg1, Arg2)

[8] SICStus Prolog supports 32 bit integers, and Quintus supports 29-bit integers. Thus, both are sufficient in this regard.

This is a data structure description. Argument names, such as *Arg1* and *Arg2*, which are referenced in the description are set in italics. By convention, arguments generally follow the order used by the corresponding Xlib structure. References to other data structures or predicates are set in italics. Any extended discussion of an argument merits its own paragraph.

xPredicate(+InputArg, ?EitherArg, -OutputArg, -NextArg)

This is a predicate description. The format is the same used with data structures with the following additions. Arguments prefixed with a plus should be instantiated when calling the predicate. Those prefixed with a minus should not be instantiated. A question mark prefix indicates that the argument may be instantiated. Violating the indicated argument mode usually just causes predicate failure. Users should note that arguments prefixed with a minus will often return information which cannot otherwise be recreated.

Also, input arguments usually come before output ones. New input arguments are added after the X defined input arguments. Argument descriptions may provide information about what values are allowed. The major styles are described below.

InputArg: This argument uses an established data type. (boolean)

EitherArg: This argument lists all of the allowed values. (xValue1, xValue2, xValue3)

OutputArg: This argument uses a data type described in the text, but also may take on some special value. (xRed for the color red)

NextArg: This argument uses a data type described in the text, but also may take on some special value. This special value is usually an X constant. (xCopyFromParent allowed)

5 Data Types

In XWIP, Prolog integers are used to represent XIDs, descriptors, atoms and boolean values. All of these are either zero or positive. XIDs represent a wide range of objects including windows, pixmaps, colormaps, cursors and fonts. For booleans, the value "true" is represented by the atom *xTrue* and "false" by *xFalse*. X time values may commonly assume values that exceed Prolog integer precision. Therefore, X time values may always be represented as a *split* structure (see below).

The generic "mask" type is represented by a list of constants (atoms). Event mask components (called "event sets") are described in the Events section. Others are described as they are introduced. Unfortunately, there are other uses of the term "mask". One use *names* a bit mask of some sort. In XWIP, the

type of such an object is an integer. Another use describes a graphics filtering operation. Users should carefully note the context in which the term is used.

Some special data structures are used by XWIP. These are detailed below. Such objects are represented in Prolog as structures. The functor of such a structure may be thought of as the data type of the object. Predicate descriptions describe where these special types may be used.

xArc(X, Y, Width, Height, StartAngle, ArcAngle)

The arc structure describes an arc in terms of a bounding box and angles. The first four elements of this structure define the bounding box of the arc in the same format used for the rectangle structure. *StartAngle* and *ArcAngle* are given in terms of $\frac{1}{64}$ th of a degree. The angles are measured according to mathematical rather than geographic conventions. A zero *StartAngle* ray is horizontal, originating from the center of the bounding box leading outward toward the right. Larger starting angles are measured counterclockwise from the zero angle. The arc, itself, sweeps through the *ArcAngle* in a counterclockwise direction.

xCell(Pixel, Color)

The cell structure represents a colormap cell and its associated color. *Pixel* is an integer which references a valid colormap cell. *Color* is a color structure.

xColor(Red, Green, Blue)

The color structure is used to represent an exact color in terms of its primary color intensities. The elements of this structure are positive Prolog integers. In a few specifically noted circumstances, the structure is used to specify what color component elements are to be changed in an operation. In these circumstances, the atom "xChange" and the atom "xNoChange" are also legitimate values for the structure elements.

xDelta(Delta)

The delta structure specifies an integer horizontal skip in a list of text drawing instructions.

xEvent(Type, Serial, SendEvent, Connection, ...)

The event structure describes an X event. The first four fields of all events are the same. Other fields are type dependent. (see the Events section)

Type: event types, one of the following constants, xKeyPress, xKeyRelease, xButtonPress, xButtonRelease, xMotionNotify, xEnterNotify, xLeaveNotify, xFocusIn, xFocusOut, xKeymapNotify, xExpose, xGraphicsExpose, xNoExpose, xVisibilityNotify, xCreateNotify, xDestroyNotify, xUnmapNotify, xMapNotify, xMapRequest, xReparentNotify, xConfigureNotify, xConfigureRequest, xGravityNotify, xResizeRequest,

xCirculateNotify, xCirculateRequest, xPropertyNotify, xSelectionClear, xSelectionRequest, xSelectionNotify, xColormapNotify, xClientMessage, or xMappingNotify.

Serial: event serial number.

SendEvent: indicates whether this was an event generated by *xSendEvent.* (boolean)

Connection: connection the event was reported on.

xFont(Font)

The font structure specifies a font change in a list of text drawing instructions.

xKeyboardMap(Map_n, Map_{n+1}, ...)

The keyboard map structure contains the keysyms associated with a range of consecutive keycodes. Each *Map* is the keysyms structure associated with that keycode.

xKeycodes($Code_1$, $Code_2$, ...)

The keycodes structure contains the keycodes associated with a particular logical modifier. The xNoSymbol atom may be used when there is no keycode for that position. (see modifier map structure)

xKeysyms(Sym_1, Sym_2, ...)

The keysyms structure contains the keysyms associated with a particular keycode. The xNoSymbol atom may be used when there is no keycap for that position. (see keyboard map structure)

xModifierMap(Map_1, ..., Map_8)

The modifier map contains the keycodes associated with the 8 modifiers. Each *Map* is a keycodes structure.

xMotion(Time, X, Y)

The motion structure describes a pointer motion event. *Time* may be a split structure. X and Y are integers.

xPoint(X, Y)

The point structure describes a particular pixel in a window. X and Y are integers.

xPointerMap($Button_1$, $Button_2$, ...)

The pointer map structure describes the mapping of physical to logical buttons. The Nth element indicates what logical button is pressed when the Nth physical button is pressed.

xProperty(Format, Data)

The property structure is generally used to represent properties that cannot be represented as Prolog atoms. *Format* is the X format of the property and should be one of the numbers: 8, 16 or 32. This represents the basic unit of the property in bits. A property representable as an atom is in format 8. Format information is used by the X server for byte swapping purposes. *Data* is a list of zero or more numbers which make up the elements of the property. Property data is considered to be unsigned. Therefore, negative Prolog integers will be interpreted by X as large positive numbers. Format 32 data elements may be specified using split structures.

xRectangle(X, Y, Width, Height)

The rectangle structure describes a rectangle in terms of its upper left location (X, Y) and its width and height. *X* and *Y* are integers. *Width* and *Height* must be non-negative.

xSegment(X1, Y1, X2, Y2)

The segment structure describes a line segment in terms of its endpoints (X1, Y1) and (X2, Y2). All components are integers.

xSplit(Most, Least)

The split structure is generally used to represent a number too large to represent as a normal Prolog integer. *Most* is a Prolog integer containing the most significant 16 bits of the number. *Least* contains the least significant 16 bits.

xText(Format, Codes)

The text structure is generally used to describe a text string in a list of text drawing instructions that cannot be represented as a Prolog atom. *Format* specifies the number of bits in each character code. Legal values are 8 or 16. *Codes* is a list of character codes. Atoms are equivalent to format 8 text.

6 Connections

For historical reasons, a network connection, in X, is called a "display". (In older versions of X, the mapping between network connections and display screens was one to one.) Here, a network connection is simply called a "connection". Within XWIP, connections are referenced by descriptors, which are integers.

xOpenConnection(+Name, -Connection)

opens a connection to the X server. If *Name* is nil, the DISPLAY environment variable is used to determine the X display opened. Otherwise, *Name* should be an atom naming an X display according to the usual X

conventions. On success, *Connection* will be instantiated to a connection descriptor. If the X connection cannot be opened or if not enough memory can be allocated for internal data structures, the predicate will fail (and a diagnostic message will be printed).

xCloseConnection(+Connection)
: closes a connection to the X server. *Connection* should be a valid connection descriptor.

A large amount of information is associated with a connection. The following is a brief overview of the structure of this data. Each connection has a number of "screens" associated with it. A screen corresponds to an actual physical display screen device. Each screen has a number of allowable "depths" associated with it. A window (or pixmap) is made up of pixels. The number of bits per pixel used in a particular window (or pixmap) is called the depth of that window (or pixmap). The depths of all windows and pixmaps on a particular screen must be chosen from the set of allowable depths. Finally, each depth has a number of "visuals" associated with it. A visual is a color model. (i.e. how pixels are mapped to physical colors)

The following predicates allow examination of the various connection data structures. Some of these predicates use lists of zero or more *queries*. A query is a structure with one argument. The functor is the attribute name and the argument is the *value* of the attribute. The value may be instantiated.

xConnections(?ConnectionList)
: gives a list of all valid connection descriptors.

xQueryConnection(+Connection, ?QueryList)
: provides information about the connection specified. The following queries are possible:

  xNetworkDescriptor(?Value)
  : the underlying network file descriptor for the connection.

  xProtocolVersion(?Value)
  : X protocol version number.

  xProtocolRevision(?Value)
  : X protocol revision number.

  xServerVendor(?Value)
  : X server vendor name.

  xImageByteOrder(?Value)
  : whether image byte order is most significant first. (boolean, xFalse indicates least significant first)

  xImageUnit(?Value)
  : image scanline unit. (in bits)

xImagePad(?Value)
image scanline pad unit. (in bits)

xImageBitOrder(?Value)
whether image scanline unit bit order is most significant first. (boolean, xFalse indicates least significant first)

xVendorRelease(?Value)
X server vendor release number.

xQueueLength(?Value)
length of event queue.

xLastEvent(?Value)
serial number of last event read, which is effectively the serial number of the last request that is known to have been processed by the X server. (can return a split structure)

xLastRequest(?Value)
serial number of last request made to the X server. (can return a split structure)

xConnectionName(?Value)
connection name.

xMotionBuffer(?Value)
motion buffer size.

xDefaultScreen(?Value)
descriptor of the default screen.

xMinKeycode(?Value)
minimum keycode supported.

xMaxKeycode(?Value)
maximum keycode supported.

xScreens(?Value)
list of descriptors for the screens associated with the connection.

xScreens(?ScreenList)
gives a list of all screen descriptors.

xQueryScreen(+Screen, ?QueryList)
provides information about the specified screen. The following queries are allowed:

xConnection(?Value)
connection descriptor associated with this screen.

xRootWindow(?Value)
root window XID.

xWidth(?Value)
screen width (in pixels).

xHeight(?Value)
screen height (in pixels).

xPhysicalWidth(?Value)
physical screen width (in millimeters).

xPhysicalHeight(?Value)
physical screen height (in millimeters).

xRootDepth(?Value)
root window depth.

xRootVisual(?Value)
descriptor for root visual.

xDefaultGC(?Value)
descriptor for default graphics context.

xDefaultColormap(?Value)
default colormap XID.

xWhitePixel(?Value)
white pixel value.

xBlackPixel(?Value)
black pixel value.

xMaxColormaps(?Value)
maximum number of resident colormaps allowed.

xMinColormaps(?Value)
minimum number of resident colormaps allowed.

xBackingStore(?Value)
available backing store support. (xNotUseful, xWhenMapped, xAlways)

xSaveUnders(?Value)
available save under support. (boolean)

xRootEventMask(?Value)
initial root window event mask.

xDepths(?Value)
list of descriptors for depths associated with the screen.

xDepths(?DepthList)
gives a list of all depth descriptors.

xQueryDepth(+Depth, ?QueryList)
provides information about the specified depth. The following queries are supported:

xDepth(?Value)
depth value.

xVisuals(?Value)
list of descriptors for visuals associated with the depth.

xVisuals(?VisualList)
gives a list of all visual descriptors.

xQueryVisual(+Visual, ?QueryList)
provides information about the specified visual. The following queries are allowed:

xClass(?Value)
color model class. (xStaticGray, xGrayScale, xStaticColor, xPseudoColor, xTrueColor, xDirectColor)

xRedMask(?Value)
red bit mask. (defined for decomposed colormaps only)

xGreenMask(?Value)
green bit mask. (defined for decomposed colormaps only)

xBlueMask(?Value)
blue bit mask. (defined for decomposed colormaps only)

xColormapBits(?Value)
log (base 2) of number of colormap entries.

xColormapEntries(?Value)
number of colormap entries.

7 Windows

Windows are referred to in XWIP by their XIDs. Some of the following predicates use lists of zero or more *attributes.* Attributes have the same structure as queries, except they must be fully instantiated.

Coordinates in a window are measured in pixels. Larger X coordinates are to the right. Larger Y coordinates are further down. The origin is the inside (if there is a border) left corner. Window dimensions (including border width) are measured in pixels. The position of a window is measured from its parent's origin to the outside, upper left corner.

xCreateWindow(+Connection, +Parent, +X, +Y, +Width, +Height, +BorderWidth, +Depth, +Class, +Visual, +AttributeList, -Window)
creates a window.

Depth: specifies the actual depth, not a depth descriptor. (xCopyFromParent or xNone allowed)

Class: specifies the window class. (xCopyFromParent, xInputOutput, xInputOnly).

Visual: specifies a visual descriptor. (xCopyFromParent allowed).

AttributeList: attribute list, any attribute usable with *xSetWindow* is allowable here. While some attributes are redundant, they are still allowed and will be set, as if, by a subsequent *xSetWindow* invocation.

xDestroyWindow(+Connection, +Window)
destroys a window and its subwindows.

xDestroySubwindows(+Connection, +Window)
destroys all subwindows of the specified window.

xQueryWindow(+Connection, +Window, ?QueryList)
provides information about the specified window. The following queries are supported:

xX(?Value)
x coordinate of the window position in the parent's coordinate system.

xY(?Value)
y coordinate of the window position in the parent's coordinate system.

xWidth(?Value)
window width.

xHeight(?Value)
window height.

xBorderWidth(?Value)
border width.

xDepth(?Value)
depth.

xRootWindow(?Value)
root window XID of screen containing this window.

xScreen(?Value)
descriptor of screen containing this window.

xVisual(?Value)
visual descriptor.

xClass(?Value)
window class. (xInputOutput, xInputOnly)

xUnionEventMask(?Value)
union of all clients' event masks for this window.

xBitGravity(?Value)
bit gravity. (xForget, xNorthWest, xNorth, xNorthEast, xWest, xCenter, xEast, xSouthWest, xSouth, xSouthEast, xStatic)

xWinGravity(?Value)
window gravity. (same as xBitGravity, except xForget is replaced by xUnmap)

xBackingStore(?Value)
backing store. (xNotUseful, xWhenMapped, xAlways)

xBackingPlanes(?Value)
backing planes.

xBackingPixel(?Value)
backing pixel.

xOverideRedirect(?Value)
override redirect. (boolean)

xSaveUnder(?Value)
save under. (boolean)

xEventMask(?Value)
event mask for this client.

xDontPropagate(?Value)
event mask for events that should not propagate.

xColormap(?Value)
colormap XID.

xColormapLoaded(?Value)
indicates whether the colormap is loaded. (boolean)

xState(?Value)
window state. (xUnmapped, xUnviewable, xViewable)

xSetWindow(+Connection, +Window, +AttributeList)
sets the attributes of the window. The attributes and their values are listed below. The following attributes are supported:

xX, xY, xWidth, xHeight, xBorderWidth, xBitGravity, xWinGravity, xBackingStore, xBackingPlanes, xBackingPixel, xOverrideRedirect, xSaveUnder, xEventMask, xDontPropagate, xColormap:
same as *xQueryWindow*.

xSibling(+Value)
sibling window XID, if a sibling window is specified the stacking mode (see below) is changed relative to this window, otherwise it is changed relative to the window stack.

xStackMode(+Value)
stacking mode. (xAbove, xBelow, xTopIf, xBottomIf, xOpposite)

xBackPixmap(+Value)
background pixmap XID. (xNone, xParentRelative allowed)

xBackPixel(+Value)
background pixel. (overrides background pixmap)

xBorderPixmap(+Value)
border pixmap XID. (xCopyFromParent allowed)

xBorderPixel(+Value)
border pixel. (overrides border pixmap)

xCursor(+Value)
cursor XID to be used in the window.

xQueryTree(+Connection, +Window, ?Root, ?Parent, ?Children)
provides information about a window's position in its screen's window tree. *Children* is a (possible empty) list of window XIDs.

xTranslateCoordinates(+Connection, +Source, +Dest, +SrcX, +SrcY, ?DestX, ?DestY, ?Child)
translates window coordinates. This predicate fails if *Source* and *Dest* windows are on different screens.

xMapWindow(+Connection, +Window)
maps the specified window.

xMapSubwindows(+Connection, +Window)
maps all subwindows of the specified window.

xUnmapWindow(+Connection, +Window)
unmaps the specified window.

xUnmapSubwindows(+Connection, +Window)
unmaps all subwindows of the specified window.

xCirculateSubwindows(+Connection, +Window, +LowerHighest)
circulates a subwindow of the specified window. *LowerHighest* is a boolean. If not set, the operation performed is RaiseLowest.

8 Atoms and Properties

The X server names some objects with atoms. These atoms are distinct from Prolog atoms. The X server uses numbers to represent atoms. In addition, X atoms also have a printable representation as a character string. In XWIP, *X atoms* are represented as Prolog integers, and *X atom names* are represented as Prolog atoms. There are a number of predefined atoms in the X server. By convention, their names are in upper case with multiple components separated by underscore. For a complete list, see the Xlib manual.

xAtomExists(+Connection, +XAtomName, ?XAtom)
succeeds, if the X atom corresponding to the specified X atom name exists in the X server. On success, *XAtom* is instantiated.

xAtom(+Connection, ?XAtomName, ?XAtom)
provides a mapping between X atom names and X atom numbers. With *XAtomName* instantiated, the corresponding X atom is unified with *XAtom*. If the X atom corresponding to *XAtomName* does not exist, it is created. If *XAtomName* is uninstantiated, the reverse mapping is provided.

X properties and their associated X types are named by atoms. Property values have two different sorts of representations in Prolog. If they are printable as normal character strings, they can be represented as Prolog atoms. Otherwise, they are represented as a *property* structure. The "format" of a property has no semantic meaning and is only used for byte swapping purposes. The "type" of a property is the logical type of the property. X places no restriction on what atom numbers can be used for types.

xWindowProperties(+Connection, +Window, ?PropertyList)
gives a list of the properties associated with the specified window.

xGetProperty(+Connection, +Window, +Property, +Offset, +Length, +DeleteIfEnd, +RequestedType, +WantAtom, ?Type, ?Remaining, ?Value)
obtains the value of an X property. If the specified property does not exist, the predicate fails. If the requested type does not match the actual type, an empty property is returned (but other information is updated properly).

Offset, Length: in terms of 32 bit units (even if the property format is not 32).

DeleteIfEnd: If set to true, and the property is successfully read and there are no elements remaining, the property is deleted. (boolean)

RequestedType: requested type of the property (xAny allowed).

WantAtom: If set to true and the property can be represented as a string, *Value* is instantiated as an atom. Otherwise, the property is represented in the property structure form. (boolean)

Type: actual type of the property.

Remaining: unread portion of the property in terms of the format units.

Value: Format 32 items may be returned using split structures.

xSetProperty(+Connection, +Window, +Property, +Type, +Mode, +Data)
changes the value of the specified property.

Mode: specifies the exact operation performed. (xReplace, xPrepend, xAppend)

Data: may be specified using split structures.

xDeleteProperty(+Connection, +Window, +Property)
deletes the specified property from the specified window.

xRotateProperties(+Connection, +Window, +Positions, +PropertyList)
rotates the specified properties in the property array.

Selections are a special type of property, explicitly used for transferring data between clients.

xGetSelectionOwner(+Connection, +Selection, ?Window)
gets the owner window of the specified selection. This predicate fails, if the specified selection is not owned.

xSetSelectionOwner(+Connection, +Selection, +Owner, +Time)
sets the owner window of the specified selection. *Owner* maybe xNone. *Time* may be xCurrentTime.

xConvertSelection(+Connection, +Selection, +DestType, +DestProperty, +DestWindow, +Time)
requests conversion of a selection. *DestProperty* may be xNone. *Time* may be xCurrentTime.

9 Pixmaps

Pixmaps are referred to in XWIP by XIDs. Pixmaps are rectangular pixel arrays. Most graphics operations work equally well on windows or pixmaps. But unlike windows, pixmaps cannot be directly mapped to the screen. However, they can be used as the template for tiling patterns, cursors, etc. Pixmaps are associated with a particular screen at creation and can only be used with that screen.

xCreatePixmap(+Connection, +Screen, +Width, +Height, +Depth, -Pixmap)
creates a pixmap. *Screen* is a screen descriptor. *Depth* is a depth value not a depth descriptor.

xDestroyPixmap(+Connection, +Pixmap)
destroys a pixmap.

xQueryPixmap(+Connection, +Pixmap, ?QueryList)
provides information about the specified pixmap. The following queries are supported:

xWidth, xHeight, xBorderWidth, xDepth, xScreen:
same as *xQueryWindow*.

10 Color

Colormaps provide the mapping between pixel values and hardware color values. The actual interpretation of the hardware color values depends on the visual (pseudo-color, gray-scale, etc.) associated with the colormap. In XWIP, colormaps are referenced by their XIDs.

Colormaps may be read-only and shared or writable and exclusive access. A colormap has no effect on the actual screen until it is "loaded" into the resident set.

xCreateColormap(+Connection, +Screen, +Visual, +AllocAll, -Colormap)
creates a colormap.

Screen: screen descriptor.

Visual: visual descriptor.

AllocAll: specifies whether the whole colormap should be preallocated as writeable. (boolean, xFalse indicates that no preallocation should be done)

xDestroyColormap(+Connection, +Colormap)
destroys a colormap.

xMoveToNewColormap(+Connection, +Current, -New)
moves and frees all entries of a client's colormap. The colormap entries are moved to a newly created colormap.

xLoadedColormaps(+Connection, +Screen, -ColormapList)
gives a list of the loaded colormaps. *Screen* is a screen descriptor.

xLoadColormap(+Connection, +Colormap)
loads a colormap into the resident set.

xUnloadColormap(+Connection, +Colormap)
removes a colormap from the resident set.

xParseColor(+Connection, +Colormap, +Specification, -ExactColor)
parses a standard X color specfication. *Specification* may use the "hex string" style or name a color in the color database.

xQueryColor(+Connection, +Colormap, +Name, -ActualColor, -ExactColor)
looks up the specified color in a colormap. *ActualColor* and *ExactColor* are color structures. (Note this is not the equivalent of the Xlib XQueryColor function.)

xGetColors(+Connection, +Colormap, ?CellList)
gives the colors associated with the specified pixels. *CellList* should have the pixel element of each cell structure instantiated. The color component of each cell may be instantiated.

xRequestColor(+Connection, +Colormap, +Color, -Cell)
requests the closest color possible in the specified colormap. *Color* is the requested color structure. *Cell* is the returned cell structure. The actual color cell is shared and read-only.

xRequestNamedColor(+Connection, +Colormap, +Name, -ActualCell, -ExactColor)
requests the closest color possible in the specified colormap. The atom *Name* specifies the name of the requested color. *ActualCell* is the actual cell provided in the colormap. The cell is shared and read-only. *ExactColor* is the exact color requested.

xAllocColorCells(+Connection, +Colormap, +Contiguous, +Planes, +Pixels, -PlaneList, -PixelList)
allocates color cell/plane combinations for PseudoColor visuals. *Contiguous* is a boolean.

xAllocColorPlanes(+Connection, +Colormap, +Contiguous, +Colors, +Reds, +Greens, +Blues, -PixelList, -RedMask, -GreenMask, -BlueMask)
allocates color cell/plane combinations for DirectColor visuals. *Contiguous* is a boolean.

xFreeColors(+Connection, +Colormap, +PixelList, +PlaneMask)
frees colors in a colormap.

PlaneMask: This argument is different depending on how the colors were obtained. If the colors were "requested", then this argument should be zero. If the colors came from *xAllocColorCells*, then this argument should be the inclusive or of all of the *Planes* returned from that predicate. In the case of *xAllocColorPlanes*, this argument should be the inclusive or of all of the *Reds*, *Greens* and *Blues* obtained.

xSetColors(+Connection, +Colormap, +CellList)
stores colors in a colormap.

CellList: a list of cell structures. The component color structures may use the xNoChange atom. A primary color intensity specified as xNoChange retains its previous value.

xSetNamedColor(+Connection, +Colormap, +Name, +Cell)
stores a named color into a colormap. The color specified by *Name* is stored into the pixel specified by *Cell*. The color component of *Cell* must use the xChange and xNoChange atoms. Any primary color intensity specified as xNoChange retains its previous value. Those specified as xChange take on the corresponding value for the named color.

11 Graphics Contexts

A graphics context can be considered a "pen" that X uses to perform graphics operations. This "pen", however, specifies more than just line width and color. It also specifies parameters for such things as clipping regions, tiling, etc. Graphics contexts are referred to by descriptors.

xGCs(?GCList)
gives a list of all graphics context descriptors.

xCreateGC(+Connection, +Drawable, +AttributeList, -GC)
creates a graphics context. *AttributeList* is any attribute list usable with *xSetGC*.

xDestroyGC(+Connection, +GC)
destroys a graphics context.

xCopyGC(+Connection, +Src, +Mask, +Dest)
copies portions of a graphics context to another graphics context.

Mask: list of graphics context components to be copied. The components are specified by giving their corresponding attribute names. Any of the attribute names listed under *xQueryGC* may be used, with the exception of xClipList and xDashList.

xQueryGC(+GC, ?QueryList)
provides information about the specified graphics context.

xFunction(?Value)
logical operation. (xClear, xAnd, xAndReverse, xCopy, xAndInverted, xNoop, xXor, xOr, xNor, xEquiv, xInvert, xOrReverse, xCopyInverted, xOrInverted, xNand, xSet)

xPlaneMask(?Value)
plane mask.

xForeground(?Value)
foreground pixel.

xBackground(?Value)
background pixel.

xLineWidth(?Value)
line width (as defined in the X Protocol Document). (xThinLine allowed)

xLineStyle(?Value)
line style. (xSolid, xOnOffDash, xDoubleDash)

xCapStyle(?Value)
line cap style. (xNotLast, xButt, xRound, xProjecting)

xJoinStyle(?Value)
line join style. (xMiter, xRound, xBevel)

xFillStyle(?Value)
fill style. (xSolid, xTiled, xStippled, xOpaqueStippled)

xFillRule(?Value)
fill rule, specifies whether the "winding" rule is used. (boolean, xFalse indicates the "even-odd" rule is used)

xTile(?Value)
tile pixmap.

xStipple(?Value)
stipple pixmap.

xTileStipXOrigin(?Value)
tile or stipple x origin.

xTileStipYOrigin(?Value)
tile or stipple y origin.

xFont(?Value)
current font XID.

xSubwindowMode(?Value)
subwindow mode, specifies whether clipping should not be performed against inferior InputOutput windows. (boolean, xFalse indicates clipping)

xGraphicsExposures(?Value)
graphics exposures, specifies whether GraphicsExpose events should be reported when *xCopyArea* or *xCopyPlane* operations are done with this graphics context. (boolean)

xClipXOrigin(?Value)
x clip origin.

xClipYOrigin(?Value)
y clip origin.

xClipMask(?Value)
clip mask. (integer)

xDashOffset(?Value)
dash offset.

xDashLength(?Value)
dash length. (same as the Xlib GCDashList component)

xArcMode(?Value)
arc mode, specifies whether "pie slice" mode should be used for *xFillArcs* operations. (boolean, xFalse indicates chord mode)

xClipList(?Value)
specifies if the clip mask is a list of rectangles. (boolean)

xDashList(?Value)
specifies if dash mode is a list. This is not the same as the Xlib GCDashList component. (boolean)

xQueryBestSize(+Connection, +Mode, +Screen, +Width, +Height, ?ReturnWidth, ?ReturnHeight)
obtains the best server dependent sizes.

Mode: specifies the exact size returned. (xCursor, xTile, xStipple)

Screen: screen descriptor.

xSetGC(+Connection, +GC, +AttributeList)
sets elements of a graphics context.

xFunction, xPlaneMask, xForeground, xBackground, xLineWidth, xLineStyle, xCapStyle, xJoinStyle, xFillStyle, xFillRule, xTile, xStipple, xTileStipXOrigin, xTileStipYOrigin, xFont, xSubwindowMode, xGraphicsExposures, xClipXOrigin, xClipYOrigin, xClipMask, xDashOffset, xDashLength, xArcMode:
same as those specified in *xQueryGC*.

xSetClips(+Connection, +GC, +XOrigin, +YOrigin, +RectangleList, +Ordering)
sets clipping rectangles for a graphics context.

RectangleList: a list of *rectangle* structures.

Ordering: clip rectangle ordering. (xUnsorted, xYSorted, xYXSorted, xYXBanded)

xSetDashes(+Connection, +GC, +DashOffset, +DashList)
sets the dash mode for a graphics context. *DashList* is a list of dash lengths.

12 Graphics

The following predicates may generate exposures.

xClearArea(+Connection, +Window, +X, +Y, +Width, +Height, +Exposures)
clears the specified rectangular area. *Exposures* is a boolean.

Width: if this value is zero, then the X position is interpreted as the window width minus *X*.

Height: if this value is zero, then the Y position is interpreted as the window height minus *Y*.

xCopyArea(+Connection, +Src, +Dest, +GC, +SrcX, +SrcY, +Width, +Height, +DestX, +DestY)
copies an area. *GC* is a graphics context descriptor.

xCopyPlane(+Connection, +Src, +Dest, +GC, +SrcX, +SrcY, +Width, +Height, +DestX, +DestY, +Plane)
copies a bit plane. *GC* is a graphics context descriptor.

Many of the following predicates take a "drawable" argument. A drawable may be either a window XID or a pixmap XID.

xDrawPoints(+Connection, +Drawable, +GC, +PointList, +Relative)
draws the specified list of points. *GC* is a graphics context descriptor. ***Relative* is a boolean. If set, coordinates are interpreted relative to the last coordinate. Otherwise, coordinates are considered to be absolute window coordinates.**

xDrawLines(+Connection, +Drawable, +GC, +PointList, +Relative)
draws the specified list of lines. *GC* is a graphics context descriptor. *Relative* is specified as in *xDrawPoints*.

xDrawSegments(+Connection, +Drawable, +GC, +SegmentList)
draws the specified list of line segments. *GC* is a graphics context descriptor.

xDrawRectangles(+Connection, +Drawable, +GC, +RectangleList)
draws the specified list of rectangles. *GC* is a graphics context descriptor.

xDrawArcs(+Connection, +Drawable, +GC, +ArcList)
draws the specified list of arcs. *GC* is a graphics context descriptor.

xFillRectangles(+Connection, +Drawable, +GC, +RectangleList)
fills the specified list of rectangles. *GC* is a graphics context descriptor.

xFillPolygon(+Connection, +Drawable, +GC, +PointList, +Shape, +Relative)
fills the specified polygon. *GC* is a graphics context descriptor. *Relative* is specified as in *xDrawPoints*.

Shape: hint to improve server performance. (xComplex, xNonconvex, xConvex)

xFillArcs(+Connection, +Drawable, +GC, +ArcList)
fills the specified list of arcs. *GC* is a graphics context descriptor.

13 Images

Images are referenced by descriptor.

xImages(?ImageList)
gives a list image descriptors.

xCreateImage(+Connection, +Visual, +Depth, +Format, +Offset, +Width, +Height, +Pad, +BytesPerLine, -Image)
creates an undefined image.

Visual: visual descriptor.

Depth: depth value, not a depth descriptor.

Format: specifies format. (xXYPixmap, xZPixmap)

Pad: scanline pad unit. The default value is the same used by the connection. (8, 16, 32, xDefault)

BytesPerLine: bytes per scanline. (xDefault for compute it from other data)

xDestroyImage(+Image)
destroys an image.

xQueryImage(+Image, ?QueryList)
provides information about the specified image.

xWidth(?Value)
width.

xHeight(?Value)
height.

xOffset(?Value)
x offset.

xFormat(?Value)
format. (xXYPixmap, xZPixmap)

xImageByteOrder(?Value)
whether byte order is most significant first. (boolean, xFalse indicates least significant first)

xImageUnit(?Value)
scanline unit.

xImagePad(?Value)
scanline pad unit.

xImageBitOrder(?Value)
whether scanline unit bit order is most significant first. (boolean, xFalse indicates least significant first)

xDepth(?Value)
depth value.

xLineBytes(?Value)
bytes per line.

xPixelBits(?Value)
bits per pixel. (for ZPixmap)

xRedMask(?Value)
red mask. (for decomposed ZPixmap)

xGreenMask(?Value)
green mask. (for decomposed ZPixmap)

xBlueMask(?Value)
blue mask. (for decomposed ZPixmap)

xGetImage(+Connection, +Drawable, +SrcX, +SrcY, +Width, +Height, +PlaneMask, +DestImage, +DestX, +DestY)
gets an image from a drawable. (This predicate uses the Xlib function XGetSubImage.)

xPutImage(+Connection, +Drawable, +GC, +Image, +SrcX, +SrcY, +DestX, +DestY, +Width, +Height)
puts an image into a drawable. *GC* is a graphics context descriptor.

xSubImage(+Image, +X, +Y, +Width, +Height, -SubImage)
extracts a part of an image from another.

xAddPixel(+Image, +Value)
adds a constant to all pixels in the specified image.

xGetPixel(+Image, +X, +Y, -Pixel)
gets the value of the specified pixel.

xSetPixel(+Image, +X, +Y, -Pixel)
sets the value of the specified pixel.

14 Text

The terminology used in XWIP is somewhat different than that used in Xlib. In order to use a font, you must "open" it. When you are done with the font you "close" it. Both of these operations cause the server to possibly load or unload font information. However, no information is loaded or unloaded on the client side. In order to examine the font information, you must explicitly do a "load" operation on the open font, which causes XWIP to actually store the information. After use, you may "unload" the font information from XWIP. Fonts are referred to by XID. Loaded fonts are referred to by descriptor.

xOpenFont(+Connection, +Name, -Font)
opens a font.

xCloseFont(+Connection, +Font)
closes a font.

xLoadedFonts(?LoadedFontList)
provides a list of loaded font descriptors.

xLoadFont(+Connection, +Font, -LoadedFont)
loads a font.

xUnloadFont(+LoadedFont)
unloads a font.

xGetFontPath(+Connection, -PathList)
returns the list of directories (atoms) to be searched by the X server for fonts.

xSetFontPath(+Connection, +PathList)
sets the list of directories to be searched by the X server for fonts. If *PathList* is nil, the default path is restored.

xListFonts(+Connection, +Pattern, +MaxNames, -FontList)
gives list of font names matching *Pattern*. The list is up to *MaxNames* in length.

The following predicates provide information about a *loaded* font and its characters. Font character codes are either 8 or 16 bits long. Characters in a 16 bit font are considered to be in a two-dimensional array. The most significant byte of the 16 bit character code is the "row" coordinate. The least significant byte is the "column" index. An 8 bit font is considered to be completely in a row zero. X text items (i.e. character strings) can be represented as Prolog atoms or as *text* structures. Atoms may only be used when the text item is made up of printable characters from an 8 bit font.

xQueryFont(+LoadedFont, ?QueryList)
provides information about a loaded font. The following queries are supported:

xConnection(?Value)
descriptor of the connection that this font was loaded from.

xFont(?Value)
font XID assigned by the connection that this font was loaded from.

xDirection(?Value)
right to left font direction hint. (boolean, xFalse indicates left to right)

xMinColumn(?Value)
minimum character column.

xMaxColumn(?Value)
maximum character column.

xMinRow(?Value)
minimum character row.

xMaxRow(?Value)
maximum character row.

xAllExist(?Value)
specifies whether all characters have nonzero bounding boxes. (boolean)

xDefaultChar(?Value)
specifies the default character to print for missing characters.

xMinLeft(?Value)
minimum left bearing of all characters.

xMinRight(?Value)
minimum right bearing of all characters.

xMinWidth(?Value)
minimum width of all characters.

xMinAscent(?Value)
minimum ascent of all characters.

xMinDescent(?Value)
minimum descent of all characters.

xMinAttribute(?Value)
minimum attribute of all characters.

xMaxLeft(?Value)
maximum left bearing of all characters.

xMaxRight(?Value)
maximum right bearing of all characters.

xMaxWidth(?Value)
maximum width of all characters.

xMaxAscent(?Value)
maximum ascent of all characters.

xMaxDescent(?Value)
maximum descent of all characters.

xMaxAttribute(?Value)
maximum attribute of all characters.

xAscent(?Value)
font ascent.

xDescent(?Value)
font descent.

xProperties(?Value)
gives a list of all properties defined on the loaded font.

xGetFontProperty(+LoadedFont, +Property, ?Value)
returns the value of a loaded font property.

xGetCharInfo(+LoadedFont, +Character, -LeftBearing, -RightBearing, -Width, -Ascent, -Descent, -Attributes)
provides character specific information.

xTextExtents(+LoadedFont, +Text, -LeftBearing, -RightBearing, -Width, -Ascent, -Descent)
gives metric information for the string using the specified font.

The following predicates draw text.

xDrawText(+Connection, +Drawable, +GC, +X, +Y, +DrawList)
draws text. Only pixels in the body of each character are modified.

GC: graphics context descriptor.

DrawList: a list of text drawing instructions. The list may contain text (either as atoms or text structures), font structures and delta structures. All text must either be 8 bit (atoms and format 8) or 16 bit (format 16) and match the type of font.

xImageText(+Connection, +Drawable, +GC, +X, +Y, +Text)
draws image text. Image text draws the character body and fills the bounding box of the character with the background color. Terminal emulators require this type of behavior.

15 Cursors

A cursor is the screen image of the pointer. A cursor is referred to by an XID. Foreground and background colors used in cursor related predicates are specified as color structures, not pixel values. Therefore, it is not required to store the color values in a colormap before using them with a cursor.

xCreateFontCursor(+Connection, +Shape, -Cursor)
creates a cursor from the standard cursor font. *Shape* can be any one of the atoms listed in Appendix B.

xCreateGlyphCursor(+Connection, +SrcFont, +MaskFont, +SrcChar, +MaskChar, +Foreground, +Background, -Cursor)
creates a cursor from the specified font information. *MaskFont* and *MaskChar* can be xNone.

xCreatePixmapCursor(+Connection, +Source, +Mask, +Foreground, +Background, +X, +Y, -Cursor)
creates a cursor from pixmap information. *Mask* can be xNone.

xDestroyCursor(+Connection, +Cursor)
destroys a cursor.

xRecolorCursor(+Connection, +Cursor, +Foreground, +Background)
changes the color of a cursor.

16 The Pointer

A pointer device can be any of a number of devices, such as a mouse, graphics tablet, track ball, etc. The pointer also has upto 5 buttons (e.g. mouse buttons). X supports only one pointer per connection.

xPointerState(+Connection, +Window, -Root, -Child, -RootX, -RootY, -WinX, -WinY, -State)
returns the current position of the pointer and state of the modifiers and buttons. This predicate fails if the pointer is not on the same screen as the specified window.

State: is a mask describing button and modifier state. The representation of *State* is a list of zero or more of the following: Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, Mod5, Button1, Button2, Button3, Button4, Button5.

xWarpPointer(+Connection, +Source, +Dest, +SrcX, +SrcY, +Width, +Height, +DestX, +DestY)
warps the position of the pointer. *Source* and *Dest* may be xNone.

Width, Height: specify the dimensions of the source rectangle containing the pointer. If *Width* is 0, it is replaced by the window width minus *SrcX*. If *Height* is 0, it is replaced by the window height minus *SrcY*.

xQueryPointer(+Connection, ?QueryList)
provides information about pointer acceleration setting. (Note this is not the equivalent of the Xlib XQueryPointer function.) The following queries are allowed:

xNumerator(?Value)
acceleration numerator.

xDenominator(?Value)
acceleration denominator.

xThreshold(?Value)
acceleration threshold.

xSetPointer(+Connection, +AttributeList)
sets pointer acceleration. The following attributes may be specified:

xNumerator, xDenominator, xThreshold:
same as in *xQueryPointer*.

xGetPointerMapping(+Connection, +Length, -Map)
gets the mapping of buttons 1 to *Length*. Meaningless values may be given for non-existent buttons. *Map* is a pointer map structure.

xSetPointerMapping(+Connection, +Map)
sets the mapping of pointer buttons. *Map* is interpreted the same as in *xGetPointerMapping*. This predicate may fail if the buttons to be changed are pressed down or if a specified button does not exist.

17 The Keyboard

The "keyboard" in X also includes the terminal bell and keyboard LEDs. X supports only one keyboard per connection.

xDownKeymap(+Connection, -Keymap)
returns the keymap of pressed down keys. *Keymap* is an ascending list of keycodes corresponding to pressed down keys.

xQueryKeyboard(+Connection, +QueryList)
gives information about the keyboard. The following queries are allowed:

xKeyClickPercent(?Value)
key click percent.

xBellPercent(?Value)
bell percent.

xBellPitch(?Value)
bell pitch.

xBellDuration(?Value)
bell duration.

xLed(?Value)
mask of lit LEDs.

xAutoRepeatMode(?Value)
specifies whether auto repeat is on. (boolean)

xRepeatKeymap(?Value)
gives the keymap of keys that have been to set to auto repeat. (Note that the global keyboard state of auto repeat is not reflected here.) The format is the same as in *xDownKeymap*.

xSetKeyboard(+Connection, +AttributeList)
sets the state of the keyboard.

xKeyClickPercent, xBellPercent, XBellPitch, xBellDuration:
same as in *xQueryKeyboard*, except that xDefault can be specified to restore the default value.

xLed(?Value)
specifies a server specific LED, or if not specified, all LEDs.

xLedMode(?Value)
specifies whether to light an LED. (boolean)

xKey(?Value)
specifies a key, or if not specified, the global auto repeat state.

xAutoRepeatMode(?Value)
specifies whether to enable auto repeat. (xFalse, xTrue, xDefault)

xGetKeyboardMapping(+Connection, +First, +Count, -KeyboardMap)
gets the keyboard mapping for the specified range of keycodes. *KeyboardMap* is a keyboard map structure. The first keysyms structure corresponds to the keysyms for keycode *First*.

xSetKeyboardMapping(+Connection, +First, +KeyboardMap)
sets the keyboard mapping for the keycodes starting from *First*. *KeyboardMap* is a keyboard map structure. All keysyms structures must have the same arity.

xGetModifierMapping(+Connection, -ModifierMap)
gets the modifier mapping. *ModifierMap* is a modifier map structure.

xSetModifierMapping(+Connection, +ModifierMap, -Busy)
sets the modifier mapping. *ModifierMap* is a modifier map structure. All keycodes structures must have the same arity. *Busy* is a boolean. The value xTrue indicates that the request failed because at least one of the modifiers to be changed was held down already.

xBell(+Connection, +Percent)
rings the bell with a modified strength. *Percent* can be any integer from -100 to 100.

The following predicates support client keyboard processing. In order to avoid making a protocol request, these predicates make use of mapping information stored in the client. This requires the client to update this information whenever there is xMappingNotify event by calling the *xRefreshMapping* predicate.

xKeycodeToKeysym(+Connection, +Keycode, +Index, -Keysym)
maps a keycode and index to a keysym. *Keysym* may be xNoSymbol.

xKeysymToKeycode(+Connection, +Keysym, -Keycode)
maps a keysym to its first keycode. *Keysym* may be xNoSymbol.

xRefreshMapping(+Event)
updates client mapping information.

18 Window Manager Support

Most of the following predicates are seldom used outside of window managers. These predicates cover five areas: connection close processing, input focus control, "grab" processing, screen saver control and server access control.

xSetSaveSet(+Connection, +Window, +Delete)
inserts or deletes a window from the connection's save set. *Delete* is a boolean. If set, the window is removed from the save set. Otherwise, the window is added to the save set.

xReparentWindow(+Connection, +Window, +Parent, +X, +Y)
reparents a window in the same screen.

xSetCloseDownMode(+Connection, +Mode)
sets the close down mode of a client. *Mode* may be xDestroy, xPermanent or xTemporary.

xKillClient(+Connection, +Resource)
forces a close down of the client which created the specified resource. *Resource* may also be xAllTemporary.

xGetInputFocus(+Connection, -Window, -Revert)
returns the current focus state.

Window: the focus window or xNone or xPointerRoot.

Revert: revert state. (xNone, xPointerRoot, xParent)

xSetInputFocus(+Connection, +Window, +Revert, +Time)
sets the input focus. *Window* and *Revert* are specified as in *xGetInputFocus*. *Time* may be xCurrentTime.

xGrabPointer(+Connection, +GrabWindow, +NormalEvents, +EventMask, +PointerRelease, +KeyboardRelease, +ConfineWindow, +Cursor, +Time, -Status)
grabs the pointer. This success of this predicate only means that the request was correct. In order to determine if the grab was successful, the *Status* value must be examined. *Time* may be xCurrentTime.

NormalEvents: specifies whether normal pointer event processing for the grabbing client occurs during the pointer grab. (boolean, xFalse, all pointer events are reported with respect to the *GrabWindow* and only pointer events specified in *EventMask* are reported)

EventMask: event mask for special event processing. (list of zero or more of the following: xButtonPress, xButtonRelease, xEnterWindow, xLeaveWindow, xPointerMotion, xPointerMotionHint, xButton1Motion, xButton2Motion, xButton3Motion, xButton4Motion, xButton5Motion, xButtonMotion, xKeymapState)

PointerRelease, KeyboardRelease: specifies whether further events of this type are to be processed normally (as opposed to deferring them) during the grab. (boolean)

ConfineWindow: the XID of the window to confine the pointer to or xNone.

Status: reports the status of a correct request. (xSucess, xAlreadyGrabbed, xInvalidTime, xNotViewable, or xFrozen)

xUngrabPointer(+Connection, +Time)
releases the grabbed pointer. *Time* may be xCurrentTime.

xSetActivePointer(+Connection, +EventMask, +Cursor, +Time)
changes active pointer grab parameters. *EventMask* is specified as described in *GrabPointer*. *Cursor* may be xNone. *Time* may be xCurrentTime.

xGrabButton(+Connection, +Button, +Modifiers, +GrabWindow, +NormalEvents, +EventMask, +PointerRelease, +KeyboardRelease, +ConfineWindow, +Cursor)
establishes a passive pointer grab, triggered by the specified circumstances.

Button: specifies the number of the button to be grabbed. (xAny allowed)

Modifiers: modifier mask. (a list of zero or more of the following: xShift, xLock, xControl, xMod1, xMod2, xMod3, xMod4, xMod5 or xAny)

NormalEvents, EventMask, PointerRelease, KeyboardRelease, ConfineWindow: same as *xGrabPointer.*

xUngrabButton(+Connection, +Button, +Modifiers, +GrabWindow)
removes a passive pointer grab trigger. *Button* and *Modifiers* are specified as in *xGrabButton*

xGrabKeyboard(+Connection, +GrabWindow, +NormalEvents, +PointerRelease, +KeyboardRelease, +Time, -Status)
grabs the keyboard. This success of this predicate only means that the request was correct. In order to determine if the grab was successful, the *Status* value must be examined. *Time* may be xCurrentTime.

NormalEvents: specifies whether normal keyboard event processing for the grabbing client occurs during the keyboard grab. (boolean, xFalse indicates all keyboard events are reported with respect to the *GrabWindow*)

PointerRelease, KeyboardRelease, Status: **same as in *xGrabPointer.***

xUngrabKeyboard(+Connection, +Time)
releases the grabbed keyboard.

xGrabKey(+Connection, +Keycode, +Modifiers, +GrabWindow, +NormalEvents, +PointerRelease, +KeyboardRelease)
establishes a passive keyboard grab, triggered by the specified circumstances. *Time* may be xCurrentTime.

Keycode: keycode for the specific key to be grabbed. (xAny allowed)

Modifiers: same as in *xGrabButton.*

NormalEvents: same as in *xGrabKeyboard.*

PointerRelease, KeyboardRelease: same as in *xGrabPointer.*

xUngrabKey(+Connection, +Keycode, +Modifiers, +GrabWindow)
removes a passive keyboard grab trigger. *Keycode* is specified as in *xGrabKey. Modifiers* is specified as in *xGrabButton.*

xAllowEvents(+Connection, +Mode, +Time)
changes pointer and keyboard event processing modes during an active grab. *Time* may be xCurrentTime.

Mode: specifies which events are released. (xAsyncPointer, xSyncPointer, xReplayPointer, xAsyncKeyboard, xSyncKeyboard, xReplayKeyboard, xAsyncBoth, xSyncBoth)

xGrabServer(+Connection)
grabs the server.

xUngrabServer(+Connection)
releases the server.

xGetScreenSaver(+Connection, ?Timeout, ?Interval, ?Blanking, ?Exposures)
gets current screen saver parameters.

Blanking: whether the screen is blanked. (xNoBlanking, xBlanking)

Exposures: whether exposures are enabled. (xNoExposures, xExposures)

xSetScreenSaver(+Connection, +Timeout, +Interval, +Blanking, +Exposures)
sets screen saver parameters.

Timeout: timeout value in seconds. (xDefault, xDisable allowed)

Interval: change interval in seconds. (xDisable allowed)

Blanking, Exposures: same as in *xGetScreenSaver*. (xDefault allowed)

xScreenSaver(+Connection, +Activate)
turns the screen saver on or off. *Activate* is boolean. The xFalse value causes the screen saver to be reset.

xQueryAccess(+Connection, -Control, -HostList)
gives information about the current state of access control. *Control* is a boolean indicating whether access control is enabled. *HostList* is the list of hostnames on the access list.

xSetAccess(+Connection, +Enable)
turns access control on or off. *Enable* is a boolean. If it is xFalse, access control is disabled.

xSetHostAccess(+Connection, +Insert, +Host)
inserts or deletes a host from the access list. *Insert* is a boolean. If it is xFalse, the specified host is deleted. *Host* is a hostname.

19 Events

Event information is represented in the event structure. The form of an event structure is given in the data types section. The interpretation of the first four fields of all events are the same. The first four elements are: *Type*, *Serial*, *SendEvent* and *Connection*. The other fields are type dependent and are described below.

xKeyPress, xKeyRelease

xEvent(..., Window, Root, Subwindow, Time, X, Y, RootX, RootY, State, Keycode, SameScreen)

Time: time when the event was generated. If it is a "send" event, xCurrentTime is allowed. (may be a split structure)

State: state of pointer buttons and modifier keys.

SameScreen: whether the root of the source window is on the same screen as the reporting window. (boolean)

xButtonPress, xButtonRelease

xEvent(..., Window, Root, Subwindow, Time, X, Y, RootX, RootY, State, Button, SameScreen)

The *Time*, *State* and *SameScreen* elements are the same as in the *xKeyPress* structure.

xMotionNotify

xEvent(..., Window, Root, Subwindow, Time, X, Y, RootX, RootY, State, IsHint, SameScreen)

The *Time*, *State* and *SameScreen* elements are the same as in the *xKeyPress* structure.

IsHint: whether the event is a "hint" event or a normal motion event. (boolean)

xEnterNotify, xLeaveNotify

xEvent(..., Window, Root, Subwindow, Time, X, Y, RootX, RootY, Mode, Detail, SameScreen, Focus, State)

The *Time*, *SameScreen* and *State* elements are the same as in the *xKeyPress* structure.

Mode: specifies whether this event is a "normal" one or due to grab processing. (xNormal, xGrab, xUngrab)

Detail: additional information about the event. (xAncestor, xVirtual, xInferior, xNonlinear, xNonlinearVirtual)

Focus: whether the reporting window is the focus window. (boolean)

xFocusIn, xFocusOut

xEvent(..., Window, Mode, Detail)

Mode: specifies whether this event is a "normal" one or due to grab processing. (xNormal, xGrab, xUngrab, xWhileGrabbed)

Detail: additional information about the event. (xAncestor, xVirtual, xInferior, xNonlinear, xNonlinearVirtual, xPointer, xPointerRoot, xNone)

xKeymapNotify
xEvent(..., Window, Keymap)
Keymap is a list of keycodes.

xExpose
xEvent(..., Window, X, Y, Width, Height, Count)

xGraphicsExpose
xEvent(..., Drawable, X, Y, Width, Height, Count, MajorCode, MinorCode)

xNoExpose
xEvent(..., Drawable, MajorCode, MinorCode)

xVisibilityNotify
xEvent(..., Window, State)

State: the new state of window visibility. (xUnobscured, xPartial, xObscured)

xCreateNotify
xEvent(..., Parent, Window, X, Y, Width, Height, BorderWidth, OverrideRedirect)

OverrideRedirect: the value of this field in the *xCreateWindow* or *xSetWindow* requests for the window in question. (boolean)

xDestroyNotify
xEvent(..., Event, Window)

xUnmapNotify
xEvent(..., Event, Window, FromConfigure)

FromConfigure: whether this event was generated because the parent window was resized and this window had a window gravity set to UnmapGravity. (boolean)

xMapNotify
xEvent(..., Event, Window, OverrideRedirect)
The *OverrideRedirect* field is the same as in *xCreateNotify* structure.

xMapRequest
xEvent(..., Parent, Window)

xReparentNotify
xEvent(..., Event, Window, Parent, X, Y, OverrideRedirect)
The *OverrideRedirect* field is the same as in *xCreateNotify* structure.

xConfigureNotify
xEvent(..., Event, Window, X, Y, Width, Height, BorderWidth, Above, OverrideRedirect)
The *OverrideRedirect* field is the same as in *xCreateNotify* structure.

Above: the XID of sibling window under the window whose state just changed. If the window is at the bottom of the stack, this value is xNone.

xConfigureRequest
xEvent(..., Parent, Window, X, Y, Width, Height, BorderWidth, Above, Detail, ValueMask)
The *Above* value is specified as in the *xConfigureNotify* structure.

Detail: stacking mode request. (xAbove, xBelow, xTopIf, xBottomIf, xOpposite)

ValueMask: a list of which requests were made. The list is made up of zero or more of: xX, xY, xWidth, xHeight, xBorderWidth, xSibling, xStackMode.

xGravityNotify
xEvent(..., Event, Window, X, Y)

xResizeRequest
xEvent(..., Window, Width, Height)

xCirculateNotify
xEvent(..., Event, Window, Bottom)

Bottom: whether the request is to place the window on the bottom of the stack. (boolean, xFalse indicates the window should be placed on top)

xCirculateRequest
xEvent(..., Parent, Window, Bottom)
The *Bottom* field is the same as in the *xCirculateNotify* structure.

xPropertyNotify
xEvent(..., Window, Atom, Time, Delete)

Time: timestamp of when the value changed. If it is a "send" event, xCurrentTime is allowed. (can return a split structure)

Delete: whether the property was deleted. (boolean, xFalse indicates a new value was assigned to the property)

xSelectionClear

xEvent(..., Window, Selection, Time)

The *Time* field is the same as in the *xPropertyNotify* structure.

xSelectionRequest

xEvent(..., Owner, Requestor, Selection, DestType, Property, Time)

Property: a property or xNone.

Time: timestamp of the *xConvertSelection* request or xCurrentTime. (can be a split structure)

xSelectionNotify

xEvent(..., Requestor, Selection, DestType, Property, Time)

The *Property* field is the same as in *xSelectionRequest* structure.

Time: timestamp of the conversion time or xCurrentTime. (can be a split structure)

xColormapNotify

xEvent(..., Window, Colormap, Modified, Load)

Colormap: colormap XID or xNone, which indicates that the colormap (associated with *Window*) was destroyed via *xDestroyColormap*.

Modified: indicates the colormap was modified. (boolean, xFalse indicates the colormap was (un)loaded, instead of being modified)

Load: If *Modified* is false, then this field specifies whether the colormap was loaded. (boolean, xFalse indicates that the colormap was unloaded)

xClientMessage

xEvent(..., Window, MessageType, Format, Data_1, ..., Data_n)

If format is 8, there are 20 data elements. There are 10 data elements for format 16 and 5 for format 32.

xMappingNotify

xEvent(..., Window, Request, First, Count)

Request: type of request. (xModifier, xKeyboard, xPointer)

Some sets of events are described by *event masks*. An event mask is a list of zero or more *event sets*. Possible event sets include: xKeyPress, xKeyRelease, xButtonPress, xButtonRelease, xEnterWindow, xLeaveWindow, xPointerMotion, xPointerMotionHint, xButton1Motion, xButton2Motion, xButton3Motion, xButton4Motion, xButton5Motion, xButtonMotion, xKeymapState, xExposure, xVisibilityChange, xStructureNotify,

xResizeRedirect, xSubstructureNotify, xSubstructureRedirect, xFocusChange, xPropertyChange, xColormapChange, and xOwnerGrabButton.

The following predicates provide event processing capabilities.

xEventsQueued(+Connection, +Mode, -Count)
gives a count of the events queued. *Mode* may be xAlready, xAfterReading or xAfterFlush.

xNextEvent(+Connection, +Remove, -Event)
returns the next event on the queue. This predicate blocks, if no event is available. If *Remove* is true, the event is removed from the queue. A non-blocking version of this predicate can be simulated by using *xGetEvent*.

xGetEvent(+Connection, +Window, +Mask, +Type, +Remove, +Block, -Event)
searches the event queue for events matching the specified criteria. At least one of *Remove* and *Block* should be true.

Window: event window or xAny.

Mask: event mask or nil for all possible event sets.

Type: event type or xAny.

Remove: the matching event is removed. (boolean)

Block: the search will block if no event is available. The predicate will fail if no matching event is available and this argument is set to xFalse. (boolean)

xSelectEvent(+Connections, +ReadFDs, +WriteFDs, +ExceptFDs, +TimeOut, -SConnections, -SReadFDs, -SWriteFDs, -SExceptFDs, -TimeOutFlag)
selects an event source. This predicate selects the next source of events from a set of connections and file descriptors. If a connection is "selected", events are ready to be read from it. If a file descriptor is "selected", then it is ready to be read or written or some exception has occurred on it. The exact interpretation depends on which list the descriptor appears in. In addition, a timeout mechanism is provided.

Connections: list of X Connections.

ReadFDs, WriteFDs, ExceptFDs: lists of file descriptors. The respective lists are read file descriptors, write file descriptors and exception file descriptors.

TimeOut: number of milliseconds. An indefinite timeout can be indicated by using xNone. (can be split structure or xNone)

SConnections: list of selected connections.

SReadFDs, SWriteFDs, SExceptFDs: list of selected file descriptors.

TimeOutFlag: whether the request timed out. (boolean)

xPutBackEvent(+Connection, +Event)
puts the specified event back on the front of the event queue.

xSendEvent(+Connection, +Window, +Propagate, +EventMask, +Event)
sends the event described to the specified window. Fails on improperly constructed events.

Window: specifies the destination window. (xPointerWindow, xInputFocus allowed)

Propagate: specifies whether the event should be allowed to propagate. (boolean)

EventMask: Clients receive this event only if selecting on an event specified in this mask.

xFlush(+Connection)
flushes the connection output buffer to the X server.

xSync(+Connection, +Discard)
flushes the connection output buffer and waits until all requests have been processed. *Discard* is a boolean.

xSynchronize(+Connection, +On)
turns synchronous mode on or off.

xGetMotionEvents(+Connection, +Window, +Start, +Stop, -MotionEventList)
gives a list of pointer motion events in between the specified times. *MotionEventList* is a list of motion structures. *Start* and *Stop* can be xCurrentTime.

20 Miscellaneous

xListExtensions(+Connection, ?ExtensionList)
lists all extensions supported by the server.

xQueryExtension(+Connection, +Name, ?Opcode, ?Event, ?Error)
provides information about the specified extension.

xNoOp(+Connection)
does a no-op protocol request, exercising the connection.

xGetDefault(+Connection, +Program, +Option, ?Default)
obtains the value of a default. *Default* is always unified with an atom, if a default is found. This predicate fails, if no default is found.

xParseGeometry(+Specification, -Width, -Height, -X, -Y, -XNegative, -YNegative)
parses a standard X geometry specification. If a particular geometry component in *Specification* is not present, its corresponding return element is xNone. *XNegative* and *YNegative* are booleans.

xGeometry(+Screen, +Specification, +PDX, +PDY, +PDW, +PDH, -X, -Y, -W, -H, -Hints)
parses a standard X geometry specification, fills in missing elements with program defaults and figures out geometry hints.

Screen: a screen descriptor.

Specification: geometry specification.

PDX, PDY, PDW, PDH: program default values for X, Y, width and height.

X, Y, W, H: final values for X, Y, width and height.

Hints: a list of geometry hints for use with *xSetNormalHints*.

xSetStandardProperties(+Connection, +Window, +Program, +IconName, +WMHints, +Command, +NormalHints)
sets the "standard" window manager properties.

Program: the program name, used to set the WM_NAME property.

IconName: name to be used with the program when iconified, used to set the WM_ICON_NAME property.

WMHints: described with *xSetWMHints*.

Command: command used to invoke the program, used to set the WM_COMMAND property. The goal used to invoke the X client should be passed here.

NormalHints: described with *xSetNormalHints*.

xSetWMHints(+Connection, +Window, +HintList)
sets the WM_HINTS property. *HintList* is an attribute list containing any of the following requests:

xInput(+Value)
whether the application relies on the window manager to set input focus. (boolean)

xInitialState(+Value)
the initial state of the application. (xDontCare, xNormal, xZoom, xIconic, xInactive)

xIconPixmap(+Value)
the icon pixmap XID.

xIconWindow(+Value)
the icon window XID.

xIconX(+Value)
the X coordinate of the icon window.

xIconY(+Value)
the Y coordinate of the icon window.

xIconMask(+Value)
the icon mask pixmap XID.

xWindowGroup(+Value)
the window group XID.

xSetNormalHints(+Connection, +Window, +HintList)
sets the WM_NORMAL_HINTS property. *HintList* is an attribute list containing any of the following requests:

xUserX(+Value)
user specified X coordinate.

xUserY(+Value)
user specified Y coordinate.

xUserWidth(+Value)
user specified width.

xUserHeight(+Value)
user specified height.

xProgramX(+Value)
program specified X coordinate.

xProgramY(+Value)
program specified Y coordinate.

xProgramWidth(+Value)
program specified width.

xProgramHeight(+Value)
program specified height.

xMinWidth(+Value)
program specified minimum width.

xMinHeight(+Value)
program specified minimum height.

xMaxWidth(+Value)
program specified maximum width.

xMaxHeight(+Value)
program specified maximum height.

xWidthIncrement(+Value)
program specified width resize increment.

xHeightIncrement(+Value)
program specified height resize increment.

xMinAspectNumerator(+Value)
program specified minimum aspect ratio numerator.

xMinAspectDenonminator(+Value)
program specified minimum aspect ratio denominator.

xMaxAspectNumerator(+Value)
program specified maximum aspect ratio numerator.

xMaxAspectDenominator(+Value)
program specified maximum aspect ratio denominator.

xOk
succeeds if XWIP has not received any X server detected errors yet. Usually, it is used after an *xSync* (see above).

xPrecision(?Bits)
If *Bits* is instantiated, then the XWIP precision threshold (in bits) is changed to the specified value. Data elements requiring greater precision will be represented as a split structure. Unsigned data will be split at a threshold of one less than specified. By default, this value should be set properly for the particular Prolog interpreter. It cannot be set to less than 16, or greater than 33. The value 33, essentially prevents use of split structures. If *Bits* is not instantiated, the current value is returned.

A Xlib Equivalents

Most predicates use the same (except for capitalization) names as their Xlib counterparts. In those cases, where different names are used, the Xlib equivalent is listed below.

XWIP Predicate	Xlib Function(s)
xAtom	XGetAtomName
	XInternAtom
xAtomExists	XInternAtom
xCloseConnection	XCloseDisplay
xCloseFont	XUnloadFont
xDestroyColormap	XFreeColormap
xDestroyCursor	XFreeCursor
xDestroyGC	XFreeGC
xDestroyPixmap	XFreePixmap
xDownKeymap	XQueryKeymap
xDrawText	XDrawText
	XDrawText16
xGetColors	XQueryColors
xGetEvent	XCheckIfEvent
	XIfEvent
	XPeekIfEvent
xGetImage	XGetSubImage
xGetProperty	XGetWindowProperty
xImageText	XDrawImageString
	XDrawImageString16
xLoadColormap	XInstallColormap
xLoadFont	XQueryFont
xLoadedColormaps	XListInstalledColormaps
xMoveToNewColormap	XCopyColormapAndFree
xNextEvent	XNextEvent
	XPeekEvent
xOpenConnection	XOpenDisplay
xOpenFont	XLoadFont
xPointerState	XQueryPointer
xQueryAccess	XListHosts
xQueryColor	XLookupColor
xQueryConnection	Display macros and functions
xQueryKeyboard	XGetKeyboardControl
xQueryPixmap	XGetGeometry

(continued on next page)

XWIP Predicate	Xlib Function(s)
xQueryPointer	XGetPointerControl
xQueryScreen	Screen macros and functions
xQueryWindow	XGetWindowAttributes
xRefreshMapping	XRefreshKeyboardMapping
xRepeatKeymap	XGetKeyboardControl
xRequestColor	XAllocColor
xRequestNamedColor	XAllocNamedColor
xScreenSaver	XForceScreenSaver
xSetAccess	XSetAccessControl
xSetActivePointer	XChangeActivePointerGrab
xSetClips	XSetClipRectangles
xSetColors	XStoreColors
xSetGC	XChangeGC
xSetKeyboard	XChangeKeyboardControl
xSetNamedColor	**XStoreNamedColor**
xSetPointer	XChangePointerControl
xSetWindow	XChangeWindowAttributes
	XConfigureWindow
xSetProperty	XChangeProperty
xTextExtents	XTextExtents
	XTextExtents16
xUnloadColormap	XUninstallColormap
xUnloadFont	XFreeFont
xWindowProperties	XListProperties

B X Font Cursors

The following are available cursor shapes in the standard cursor font.

num_glyphs	X_cursor
arrow	based_arrow_down
based_arrow_up	boat
bogosity	bottom_left_corner
bottom_right_corner	bottom_side
bottom_tee	box_spiral
center_ptr	circle
clock	coffee_mug
cross	cross_reverse
crosshair	diamond_cross
dot	dotbox
double_arrow	draft_large
draft_small	draped_box
exchange	fleur
gobbler	gumby
hand1	hand2
heart	icon
iron_cross	left_ptr
left_side	left_tee
leftbutton	ll_angle
lr_angle	man
middlebutton	mouse
pencil	pirate
plus	question_arrow
right_ptr	right_side
right_tee	rightbutton
rtl_logo	sailboat
sb_down_arrow	sb_h_double_arrow
sb_left_arrow	sb_right_arrow
sb_up_arrow	sb_v_double_arrow
shuttle	sizing
spider	spraycan
star	target
tcross	top_left_arrow
top_left_corner	top_right_corner
top_side	top_tee
trek	ul_angle
umbrella	ur_angle
watch	xterm

Index

event masks, 38
event types, 5

xAddPixel, 24
xAllocColorCells, 18
xAllocColorPlanes, 18
xAllowEvents, 33
xArc, 5
xAtom, 15
xAtomExists, 15
xBell, 31
xButtonPress, 35
xButtonRelease, 35
xCell, 5
xCirculateNotify, 37
xCirculateRequest, 37
xCirculateSubwindows, 14
xClearArea, 21
xClientMessage, 38
xCloseConnection, 8
xCloseFont, 25
xColor, 5
xColormapNotify, 38
xConfigureNotify, 37
xConfigureRequest, 37
xConnections, 8
xConvertSelection, 16
xCopyArea, 22
xCopyGC, 19
xCopyPlane, 22
xCreateColormap, 17
xCreateFontCursor, 27
xCreateGC, 19
xCreateGlyphCursor, 28
xCreateImage, 23
xCreateNotify, 36
xCreatePixmap, 16
xCreatePixmapCursor, 28
xCreateWindow, 11
xDeleteProperty, 16
xDelta, 5
xDepths, 10
xDestroyColormap, 17
xDestroyCursor, 28
xDestroyGC, 19
xDestroyImage, 23
xDestroyNotify, 36
xDestroyPixmap, 16
xDestroySubwindows, 12
xDestroyWindow, 12
xDownKeymap, 29
xDrawArcs, 22
xDrawLines, 22
xDrawPoints, 22
xDrawRectangles, 22
xDrawSegments, 22
xDrawText, 27
xEnterNotify, 35
xEvent, 5
xEventsQueued, 39
xExpose, 36
xFillArcs, 22
xFillPolygon, 22
xFillRectangles, 22
xFlush, 40
xFocusIn, 35
xFocusOut, 35
xFont, 6
xFreeColors, 18
xGCs, 19
xGeometry, 41
xGetCharInfo, 27
xGetColors, 17
xGetDefault, 40
xGetEvent, 39
xGetFontPath, 25
xGetFontProperty, 27
xGetImage, 24
xGetInputFocus, 31
xGetKeyboardMapping, 30
xGetModifierMapping, 30
xGetMotionEvents, 40

xGetPixel, 24
xGetPointerMapping, 29
xGetProperty, 15
xGetScreenSaver, 34
xGetSelectionOwner, 16
xGrabButton, 32
xGrabKey, 33
xGrabKeyboard, 33
xGrabPointer, 32
xGrabServer, 34
xGraphicsExpose, 36
xGravityNotify, 37
xImages, 23
xImageText, 27
xKeyboardMap, 6
xKeycodes, 6
xKeycodeToKeysym, 31
xKeymapNotify, 36
xKeyPress, 35
xKeyRelease, 35
xKeysyms, 6
xKeysymToKeycode, 31
xKillClient, 31
xLeaveNotify, 35
xListExtensions, 40
xListFonts, 25
xLoadColormap, 17
xLoadedColormaps, 17
xLoadedFonts, 25
xLoadFont, 25
xMapNotify, 36
xMappingNotify, 38
xMapRequest, 36
xMapSubwindows, 14
xMapWindow, 14
xModifierMap, 6
xMotion, 6
xMotionNotify, 35
xMoveToNewColormap, 17
xNextEvent, 39
xNoExpose, 36
xNoOp, 40
xOk, 43
xOpenConnection, 7
xOpenFont, 25
xParseColor, 17
xParseGeometry, 41
xPoint, 6
xPointerMap, 6
xPointerState, 28
xPrecision, 43
xProperty, 7
xPropertyNotify, 37
xPutBackEvent, 40
xPutImage, 24
xQueryAccess, 34
xQueryBestSize, 21
xQueryColor, 17
xQueryConnection, 8
xQueryDepth, 10
xQueryExtension, 40
xQueryFont, 25
xQueryGC, 19
xQueryImage, 23
xQueryKeyboard, 29
xQueryPixmap, 16
xQueryPointer, 28
xQueryScreen, 9
xQueryTree, 14
xQueryVisual, 11
xQueryWindow, 12
xRecolorCursor, 28
xRectangle, 7
xRefreshMapping, 31
xReparentNotify, 36
xReparentWindow, 31
xRequestColor, 18
xRequestNamedColor, 18
xResizeRequest, 37
xRotateProperties, 16
xScreens, 9
xScreenSaver, 34
xSegment, 7
xSelectEvent, 39
xSelectionClear, 38
xSelectionNotify, 38
xSelectionRequest, 38
xSendEvent, 40

xSetAccess, 34
xSetActivePointer, 32
xSetClips, 21
xSetCloseDownMode, 31
xSetColors, 18
xSetDashes, 21
xSetFontPath, 25
xSetGC, 21
xSetHostAccess, 34
xSetInputFocus, 32
xSetKeyboard, 30
xSetKeyboardMapping, 30
xSetModifierMapping, 30
xSetNamedColor, 18
xSetNormalHints, 42
xSetPixel, 24
xSetPointer, 29
xSetPointerMapping, 29
xSetProperty, 15
xSetSaveSet, 31
xSetScreenSaver, 34
xSetSelectionOwner, 16
xSetStandardProperties, 41
xSetWindow, 13
xSetWMHints, 41
xSplit, 7
xSubImage, 24
xSync, 40
xSynchronize, 40
xText, 7
xTextExtents, 27
xTranslateCoordinates, 14
xUngrabButton, 33
xUngrabKey, 33
xUngrabKeyboard, 33
xUngrabPointer, 32
xUngrabServer, 34
xUnloadColormap, 17
xUnloadFont, 25
xUnmapNotify, 36
xUnmapSubwindows, 14
xUnmapWindow, 14
xVisibilityNotify, 36
xVisuals, 11
xWarpPointer, 28
xWindowProperties, 15