The Python X Library

Peter Liljenberg

Table of Contents

Introduction **1**

1 Basic X Concepts **3**

2 Package Layout **5**

3 Connect to a Display **7**

3.1 Unix Display Names 7
3.2 OpenVMS Display Names 8

4 Error Handling **9**

4.1 X Error Classes 9
4.2 CatchError 10

5 Event Handling **13**

5.1 Getting Events 13
5.2 Selecting Events 14
5.3 Event Types 16
5.4 Sending Events 26

6 X Objects **29**

6.1 Display 29
6.2 Resource 32
6.3 Colormap 32
6.4 Cursor 33
6.5 Drawable 33
6.6 Window 35
6.7 Pixmap 38
6.8 Fontable 39
6.9 GC 39
6.10 Font 39

Introduction

The Python X Library is a complete library for writing X Windows client applications in Python. It implements the entire X11R6 protocol and some extensions, and also some supplementary functionality such as a resource database. It is written entirely in Python, and is therefore entirely independent of the traditional C Xlib; it only requires a Python installation with network support.

Although the X11R6 protocol is the basis of all X client applications, it is a little too low-level to be useful for writing advanced graphical interfaces. To this end, many different widget sets has been developed, a few of the more popular are Motif, Qt and Gtk. The Python X Library is not compatible with any of these, as they are based on the C Xlib. As a result the Python X Library isn't in itself very useful for writing GUIs, but is eminent for writing small X applications with no advanced GUI, e.g. simple monitor programs, or for that matter more advanced background applications such as window managers.

1 Basic X Concepts

Here you might find an intruduction to X concepts sometime in the future. For now, I just refer to the introduction parts of the standard X documentation. A vast collection of X documentation links can be found at `http://www.rahul.net/kenton/xsites.html`.

2 Package Layout

The Python X Library is distrubuted as a Python package, named `Xlib`. It contains a number of modules providing the main library interface:

`display`
: Contains the `Display` class, providing the X connection and access to all other X objects and operations.

`error`
: Contains all `Xlib` exception classes.

`threaded`
: Imported to set up the library to be thread-safe.

`rdb`
: Contains `ResourceDB`, an X resource database class, and related code.

`X`
: Contains symbolic names for all basic X protocol constants.

`Xutil`
: Contains additional constants for mainly the ICCCM standard.

`XK`
: Contains symbolic names for all X keysyms.

`Xatom`
: Contains symbolic names for all standard X atoms.

`Xcursorfont`
: Contains symbolic names for all standard X fonts.

There is also a number of subpackages, but they will mainly be of interest to developers of the Xlib itself:

`protocol`
: The low-level protocol implementation. Contains definitions of all X request, responses, events and errors, in addition to the core network code. The high-level object oriented interface provided by `Xlib.display` is merely a wrapper around this package.

`support`
: Miscellaneous supporting code for handling threads and operating system-specific features.

`xobject`
: Contains classes for all X objects, which are returned by various display operations.

`ext`
: Contains various X protocol extensions. Extensions supported by the server are loaded automatically at the display connection.

3 Connect to a Display

To be able to perform any X operations, you must first establish a connection to the display. This is done by instantiating a `display.Display` object:

Display ([*displayname*]) Class

Create a connection to the display called *displayname*. If *displayname* is None or not provided, connect to the default display.

If the connection fails for some reason, one of the errors from the following error class tree is raised:

```
Exception
 \_error.DisplayError
   \_error.DisplayNameError
   \_error.DisplayConnectionError
```

`error.DisplayNameError` is raised if *displayname*, or the default display name, is malformed. `error.DisplayConnectionError` is raised if the connection to the X server fails. This might be caused by the network connection to the display failing, or if the client isn't authorized to connect to the display.

The syntax of a display name and how the library finds the default display name depends on the operating system where the client runs.

3.1 Unix Display Names

A display name on Unix has the following syntax:

```
[hostname]:displayno[.screenno]
```

If *hostname* is omitted this refers to a display on the same machine as the client is running on. A Unix socket is used to connect to the display identified by `displayno`, which must be an integer from 0 and upwards.

If *hostname* is specified, it gives the network name or IP-number of the machine to contact. A TCP socket is used, connecting to port 6000+*displayno* on *hostname*.

If the display contains more than one screen, *screenno* can be specified to make another screen than 0 the default screen. If the number is larger than the available maximum screen number, it will be set to the maximum.

A few examples:

```
:0           Display 0 on this host, connects to the Unix socket
             /tmp/.X11-unix/X0
:0.1         Same as above, but specifying a different default screen

myhost:0     Display 0 on myhost, connects to TCP port 6000
myhost:10    Display 10 on myhost, connects to TCP port 6010

localhost:0  Display 0 on localhost, i.e. the same host as the client
             runs on.  Uses a TCP socket, while :0 uses a Unix socket
```

The default display name is stored in the environmental variable `DISPLAY`. If that isn't set, `error.DisplayNameError` is raised.

3.2 OpenVMS Display Names

Currently the X Python Library only supports TCP connections on OpenVMS. Display names are identical to Unix display names, but if the hostname is omitted, `localhost` is used. The default display name is hard-coded to `localhost:0.0`.

In the future, the native DECWindows system should be used. This means using LOCAL or DECNET transports if set, and using the logical name `DECW$DISPLAY` as the default display.

4 Error Handling

If an X object method generates an error, it will be handled in one of two different ways depending on the kind of method. Errors are represented by X error objects in the Xlib.

If the method *does not* return data, the error will most likely be detected after the method has returned. All methods which does not return data has a parameter called `onerror`, which can be used to provide a error handler for errors generated by this method.

The error handler is called with to arguments: the error object and the low-level request object. The error *must not* do call any X object methods. If that is necessary, the error handler must store the error away for later retrieval. The class `error.CatchError` is provided for this purpose.

If no error handler is given when calling a method which generates an error, the error will be passed to the default error handler. If no default error handler is specified, the error is simply printed on `sys.stderr`.

If the method *does* return data, the error will make it impossible for it to return any valid data. An exception is raised with the error object as the exception value. It is not passed to any error handlers.

4.1 X Error Classes

X errors are structured in the following class hierarchy:

```
Exception
 \_ error.XError
     \_ error.BadRequest
     \_ error.BadValue
     \_ error.BadAtom
     \_ error.BadMatch
     \_ error.BadAccess
     \_ error.BadAlloc
     \_ error.BadName
     \_ error.BadLength
     \_ error.BadImplementation
     \_ error.XResourceError
         \_ error.BadWindow
         \_ error.BadPixmap
         \_ error.BadCursor
         \_ error.BadFont
         \_ error.BadDrawable
         \_ error.BadColor
         \_ error.BadGC
         \_ error.BadIDChoice
```

All error objects has the following attributes:

`code`

The numeric error code

`sequence_number`

The sequence number of the failed request

`resource_id`
: The bad resource id. For all the `error.XResourceError` this is a X resource object. For the other errors it is an integer, which for some errors might have no meaning

`major_opcode`
: The major opcode for the failed request

`minor_opcode`
: The minor opcode for the failed request. This will be zero for all base X11R6 request, but will be interesting for extension requests

4.2 CatchError

`error.CatchError` is an object which can be used as an error handler. It collects an error matching any of the specified types, which can be retrieved later. If several errors occures, only the last is remembered.

CatchError (**errors*) Class

Create a new error handler object. Initialize by providing all error classes you are interested in as arguments. If no error classes are provided at all, this means that all errors will be considered.

Pass the `error.CatchError` object as the `onerror` parameter to X object methods. If these methods generated any errors matching the ones specified, it can be retrieved with the following functions:

get_error ( ) Method on `CatchError`

Return the last error object caught, or None if no matching errors has occured.

get_request ( ) Method on `CatchError`

Return the request object for the last error caught, or None if no matching errors has occured.

`error.CatchError` objects can be reused:

reset ( ) Method on `CatchError`

Forget any caught error.

Since the X protocol is mostly asynchronous any error we're watching for might not have been recieved when we call `get_error`. To make sure that the request has been processed by the server and any error generated has been received by the Xlib, we must synchronize with the server.

An example of using `error.CatchError`:

```
# Resize and the foo window
# If it has been destroyed since we looked at it the last time,
# reset variable foo to None

# Create a error handler for BadWindow errors
ec = error.CatchError(error.BadWindow)

# Perform the operation
foo.configure(width = 100, height = 200, onerror = ec)

# Sync communication with server
display.sync()

# And check if there was any error
if ec.get_error():
    foo = None
```

5 Event Handling

Events are sent from the X server to the X client. Most of the event types deal with user input, but there are also event types used for inter-client communication.

Most X applications are built around an event loop, where the client waits for the server to send events. The client responds to the events, typically by doing one or more X requests.

5.1 Getting Events

Events can be sent at any time, not necessarily when the client is ready to recieve an event. Therefore they must be stored temporarily from that they are read from the network until the client is ready to handle them. Read but unhandled events are stored on an event queue in the Display object. There are two functions to access this queue:

next_event ( ) Method on `Display`

Return the next event in the event queue. If the event queue is empty, block until an event is read from the network, and return that one.

pending_events ( ) Method on `Display`

Return the number of events which can be returned without blocking.

A trivial event loop would simply loop infinitely, waiting for an event and then handling it. It could look like this:

```
while 1:
   event = disp.next_event()
   handle_event(event)
```

However, most applications need more control, e.g. to simultaneously handle a network connection or at regular intervals schedule timeouts. The module `select` is often used for this. `Display` objects can be used with `select`, since they have the required `fileno()` method. When `select` indicates that a `Display` object is ready for reading, that means that the server has sent some data to the client. That alone doesn't guarantee that an entire event has arrived, so one must first use `pending_events()` to make sure that `next_event()` will return without blocking. A simple event loop which waits for events or a one-second timeout looks like this:

```
while 1:
    # Wait for display to send something, or a timeout of one second
    readable, w, e = select.select([disp], [], [], 1)

    # if no files are ready to be read, it's an timeout
    if not readable:
        handle_timeout()

    # if display is readable, handle as many events as have been recieved
    elif disp in readable:
        i = disp.pending_events()
        while i > 0:
            event = disp.next_event()
```

```
            handle_event(event)
            i = i - 1

    # loop around to wait for more things to happen
```

5.2 Selecting Events

To avoid flooding the clients with events in which they have no interest, they must explicitly tell the server which events they are interested in. This is done by providing the `event_mask` attribute when creating windows with `Window.create_window` or in calls to `Window.change_attributes`. The value of this attribute is a mask of all types of events the client is interested in on that particular window. Whenever the server generates an event for a window which matches the clients event mask on that window, the event will be sent to the client.

The following table lists all event masks and the corresponding event types and classes. All event masks and types are integer constants defined in the module `Xlib.X`. Classes have the same name as the event type, and are defined in `Xlib.protocol.event`.

Mask	**Type and Class**	**Generated when**
ButtonMotionMask Button1MotionMask Button2MotionMask Button3MotionMask Button4MotionMask Button5MotionMask	MotionNotify	Pointer moved with any or a certain button down
ButtonPressMask	ButtonPress	Pointer button pressed
ButtonReleaseMask	ButtonRelease	Pointer button released
ColormapChangeMask	ColormapNotify	Colormap changed or installed
EnterWindowMask	EnterNotify	Pointer enters window
LeaveWindowMask	LeaveNotify	Pointer leaves window
ExposureMask	Expose NoExpose	Window needs to be redrawn
FocusChangeMask	FocusIn FocusOut	Focus changes
KeymapStateMask	KeymapNotify	After EnterNotify and FocusIn
KeyPressMask	KeyPress	Key is pressed

KeyReleaseMask	ReleasePress	Key is released
PointerMotionMask	MotionNotify	Pointer is moved
PropertyChangeMask	PropertyNotify	Window properties change
StructureNotifyMask	CirculateNotify ConfigureNotify DestroyNotify GravityNotify MapNotify ReparentNotify UnmapNotify	Window structure changes
SubstructureNotifyMask	CirculateNotify ConfigureNotify CreateNotify DestroyNotify GravityNotify MapNotify ReparentNotify UnmapNotify	Child window structure changes
ResizeRedirectMask	ResizeRequest	Controlling window size change
SubstructureRedirectMask	CirculateRequest ConfigureRequest MapRequest	Controlling changes to child windows
VisibilityChangeMask	VisibilityNotify	Window is obscured or visible

There are also some event types which are always sent to the clients, regardless of any event masks:

Type and Class	Cut'n'pasting between windows
ClientMessage	Other client sends message
MappingMotify	Keyboard mapping changes
SelectionClear SelectionNotify	Cut'n'pasting between windows
SelectionRequest	

5.3 Event Types

This section describes all event types by listing their fields and basic information on when they are generated. All events are defined in the module `Xlib.protocol.event`. All event types have the following two attributes:

type — Instance Variable of Event

Stores the X type code of this event. Type codes are integers in the range 2-127, and are defined with symbolic names in `Xlib.X`. The symbolic names are the same as the event class names, except for the special event `AnyEvent`.

send_event — Instance Variable of Event

This attribute is normally 0, meaning that the event was generated by the X server. It is set to 1 if this event was instead sent from another client.

Event object can be created by instantiating the corresponding event class, providing all the attributes described for the event as keyword arguments.

KeyPress — Event
KeyRelease — Event
ButtonPress — Event
ButtonRelease — Event
MotionNotify — Event

These events are generated when a key or a button logically changes state, or when the pointer logically moves. In the discussion below, the *source window* is the window that the event occured in. The event may be generated on some other window than the source window, see XKeyEvent(3X11) for details. The same man page also describes `MotionNotify`.

time — Instance Variable of KeyButtonPointerEvent

The server X time when this event was generated.

root — Instance Variable of KeyButtonPointerEvent

The root window which the source window is an inferior of.

window Instance Variable of KeyButtonPointerEvent
The window the event is reported on.

same_screen Instance Variable of KeyButtonPointerEvent
Set to 1 if `window` is on the same screen as `root`, 0 otherwise.

child Instance Variable of KeyButtonPointerEvent
If the source window is an inferior of `window`, `child` is set to the child of `window` that is the ancestor of (or is) the source window. Otherwise it is set to `X.NONE`.

root_x Instance Variable of KeyButtonPointerEvent
root_y Instance Variable of KeyButtonPointerEvent
The pointer coordinates at the time of the event, relative to the root window.

event_x Instance Variable of KeyButtonPointerEvent
event_y Instance Variable of KeyButtonPointerEvent
The pointer coordinates at the time of the event, relative to `window`. If `window` is not on the same screen as `root`, these are set to 0.

state Instance Variable of KeyButtonPointerEvent
The logical state of the button and modifier keys just before the event.

detail Instance Variable of KeyButtonPointerEvent
For `KeyPress` and `KeyRelease`, this is the keycode of the event key.

For `ButtonPress` and `ButtonRelease`, this is the button of the event.

For `MotionNotify`, this is either `X.NotifyNormal` or `X.NotifyHint`.

EnterNotify Event
LeaveNotify Event
If pointer motion or window hierarchy change causes the pointer to be in another window than before, these events are generated instead of a `MotionNotify` event.

The semantics are quite complex, see XCrossingEvent(3X11) for details.

time Instance Variable of EnterLeaveEvent
The server X time when this event was generated.

root Instance Variable of EnterLeaveEvent
The root window of the pointer at the end of this event.

window Instance Variable of EnterLeaveEvent
The window the event is reported for.

child Instance Variable of EnterLeaveEvent
In a `LeaveNotify` event, if a child of `window` contains the initial pointer position, this is set to that child window. In a `EnterNotify` event, if a child of `window` contains the final pointer position, this is set to that child window. Otherwise this is set to `X.NONE`.

root_x Instance Variable of EnterLeaveEvent
root_y Instance Variable of EnterLeaveEvent

The final pointer position relative to `root`.

event_x Instance Variable of EnterLeaveEvent
event_y Instance Variable of EnterLeaveEvent

The final pointer position relative to `window`.

state Instance Variable of EnterLeaveEvent

The modifier and button state at the time of the event.

mode Instance Variable of EnterLeaveEvent

One of `X.NotifyNormal`, `X.NotifyGrab` or `X.NotifyUngrab`.

detail Instance Variable of EnterLeaveEvent

One of `X.NotifyAncestor`, `X.NotifyVirtual`, `X.NotifyInferior`, `X.NotifyNonlinear`, or `X.NotifyNonlinearVirtual`.

flags Instance Variable of EnterLeaveEvent

If bit 0 is set, `window` is the focus window or an inferior of it. If bit 1 is set, `window` is on the same screen as `root`.

FocusIn Event
FocusOut Event

These events are generated when the focus changes. This is also very complex events, see XFocusChangeEvent(3X11) for details.

window Instance Variable of FocusEvent

The window the event is generated for.

mode Instance Variable of FocusEvent

One of `X.NotifyNormal`, `X.NotifyWhileGrabbed`, `X.NotifyGrab`, or `X.NotifyUngrab`.

detail Instance Variable of FocusEvent

One of `X.NotifyAncestor`, `X.NotifyVirtual`, `X.NotifyInferior`, `X.NotifyNonlinear`, `X.NotifyNonlinearVirtual`, `X.NotifyPointer`, `X.NotifyPointerRoot`, or `X.NONE`.

KeymapNotify Event

This event is generated immediately after every `EnterNotify` and `FocusIn`.

data Instance Variable of KeymapNotify

A list of 31 eight-bit integers, as returned by query_keymap.

Expose Event

This event is generated when regions of a window has to be redrawn. The regions are decomposed into a set of rectangles, and an `Expose` event is generated for each rectangle.

window Instance Variable of Expose

The window containing regions to redraw.

x Instance Variable of Expose
y Instance Variable of Expose
width Instance Variable of Expose
height Instance Variable of Expose

The coordinates of the rectangle to redraw.

count Instance Variable of Expose

At least this many more `Expose` events will immediately follow for this window. If this is the last event, `count` is set to 0.

This allows a simple application to ignore all `Expose` events with a non-zero `count`, and then redraw the entire window when the zero event is received.

GraphicsExpose Event
NoExpose Event

These events may be generated for drawables when a graphics context with `graphics_exposures` selected is used. See XGraphicsExposeEvent(3X11) for details.

Both events have these attributes:

drawable Instance Variable of GraphicsExpose

The drawable of the event.

major_event Instance Variable of GraphicsExpose
minor_event Instance Variable of GraphicsExpose

The major and minor number of the request that generated this event. For the core protocol `major_event` is always 62 (CopyArea) or 63 (CopyPlane), and `minor_event` is always 0.

GraphicsExpose have these additional attributes:

x Instance Variable of GraphicsExpose
y Instance Variable of GraphicsExpose
width Instance Variable of GraphicsExpose
height Instance Variable of GraphicsExpose

The coordinates of the event region.

count Instance Variable of GraphicsExpose

At least this many more `GraphicsExposure` events follows. The last one has `count` set to 0.

VisibilityNotify Event

These events are generated when the visibility of a viewable window is changed. See XVisibilityNotifyEvent(3X11).

window Instance Variable of VisibiltyNotify

The window of the event.

state Instance Variable of VisibiltyNotify

One of `X.VisibilityUnobscured`, `X.VisibilityPartiallyObscured`, or `X.VisibilityFullyObscured`.

CreateNotify Event

This event is generated when a window is created. `X.SubstructureNotifyMask` must be selected on the parent of the new window to receive this event.

parent Instance Variable of CreateNotify

The parent of the new window.

window Instance Variable of CreateNotify

The new window.

x Instance Variable of CreateNotify
y Instance Variable of CreateNotify
width Instance Variable of CreateNotify
height Instance Variable of CreateNotify
border_width Instance Variable of CreateNotify
override Instance Variable of CreateNotify

These values are fetched from the `create_window` call.

DestroyNotify Event

This event is generated when a window is destroyed.

event Instance Variable of DestroyNotify

The window that the event is generated for.

window Instance Variable of DestroyNotify

The window that was destroyed.

UnmapNotify Event

This event is generated when a window is unmapped.

event Instance Variable of UnmapNotify

The window that the event is generated for.

window Instance Variable of UnmapNotify

The window that was unmapped.

from_configure Instance Variable of Event

This is true if `window` was unmapped because its parent was resized and `window` had win-gravity `X.UnmapGravity`.

MapNotify Event

This event is generated when a window is mapped.

event Instance Variable of MapNotify

The window that the event is generated for.

window Instance Variable of MapNotify

The window that was mapped.

override Instance Variable of Event

This is set from the corresponding window attribute.

MapRequest Event

This event is reported to the client that has set `X.SubstructureRedirectMask` on a window, and an unmapped child of that window attempts to be mapped by some other client, unless the child has override-redirect set.

parent Instance Variable of MapRequest

The parent window.

window Instance Variable of MapRequest

The child window that attempts to be mapped.

ReparentNotify Event

This event is reported to clients selecting `X.SubstructureNotifyMask` on either the old or the new parent and to clients selecting `X.StructureNotifyMask` on the reparented window.

event Instance Variable of ReparentNotify

The window the event is generated for.

window Instance Variable of ReparentNotify

The reparented window.

parent Instance Variable of ReparentNotify

The new parent window.

x Instance Variable of ReparentNotify

y Instance Variable of ReparentNotify

The coordinates of the upper-left outer corner of `window` in `parent`.

override Instance Variable of ReparentNotify

This is set from the corresponding attribute on `window`.

ConfigureNotify Event

This event is generated when a configure request actually changes the state of the window.

event Instance Variable of ConfigureNotify

The window that the event is generated for.

window Instance Variable of ConfigureNotify

The window that has been changed.

x Instance Variable of ConfigureNotify
y Instance Variable of ConfigureNotify
width Instance Variable of ConfigureNotify
height Instance Variable of ConfigureNotify
border_width Instance Variable of ConfigureNotify

The new coordinate and geometry of `window`.

above_sibling Instance Variable of ConfigureNotify

This is `X.NONE` if this window is at the bottom of the window stack. Otherwise it is the sibling window that `window` is immediately above.

override Instance Variable of ConfigureNotify

This is set from the corresponding attribute on `window`.

ConfigureRequest Event

This event is reported to the client that has set `X.SubstructureRedirectMask` on the parent of a window that another client attempts to configure, unless the window has override-redirect set.

parent Instance Variable of ConfigureRequest

The parent of the window being configured.

window Instance Variable of ConfigureRequest

The window being configured.

value_mask Instance Variable of ConfigureRequest

A bitmask indicating which values that the caller tries to change.

x Instance Variable of ConfigureRequest
y Instance Variable of ConfigureRequest
width Instance Variable of ConfigureRequest
height Instance Variable of ConfigureRequest
border_width Instance Variable of ConfigureRequest

The window geometry in the configure call. If `X.CWX`, `X.CWY`, `X.CWWidth`, `X.CWHeight`, or `X.CWBorderWidth` is set in `value_mask`, the corresponding attributes contains the new value as given in the configure call, otherwise the current value is used.

stack_mode Instance Variable of ConfigureRequest

If `X.CWStackMode` is set in `value_mask`, this is the stack mode specified in the configure call, one of `X.Above`, `X.Below`, `X.TopIf`, `X.BottomIf`, or `X.Opposite`. If the flag is not set, this is set to `X.Above`.

sibling Instance Variable of ConfigureRequest

If `X.CWSibling` is set in `value_mask`, this is the sibling window specified in the configure call. If the flag is not set, this is set to `X.NONE`.

GravityNotify Event

This event is generated when a window is moved because its parent's size has changed.

event Instance Variable of GravityNotify

The window the event is generated for.

window Instance Variable of GravityNotify

The window that moved.

x Instance Variable of GravityNotify
y Instance Variable of GravityNotify

The new coordinates of the upper-left outer corner of `window`.

ResizeRequest Event

This is reported to the client selecting `X.ResizeRedirectMask` on a window, and another client attempts to resize it.

window Instance Variable of ResizeRedirect

The window that was attempted to be resized.

width Instance Variable of ResizeRedirect
height Instance Variable of ResizeRedirect

The requested size of the window, excluding the border.

CirculateNotify Event

This event is generated when a window is restacked caused by a circulate call.

event Instance Variable of Event
: The window the event is reported on.

window Instance Variable of Event
: The window that has been restacked.

place Instance Variable of Event
: Either `X.PlaceOnTop` or `X.PlaceOnBottom`.

CirculateRequest Event
: This event is reported to the client that has set `X.SubstructureRedirectMask` on the parent of a window that needs to be restacked as a result of a circulate call on the parent.

parent Instance Variable of CirculateRequest
: The parent window.

window Instance Variable of CirculateRequest
: The window that should be restacked.

place Instance Variable of CirculateRequest
: Where `window` should be placed, either `X.PlaceOnTop` or `X.PlaceOnBottom`.

PropertyNotify Event
: This event is generated when a property on a window is changed.

window Instance Variable of Event
: The window which the property is or was set on.

atom Instance Variable of Event
: The atom identifying the property.

time Instance Variable of Event
: The server X time when the property was changed.

state Instance Variable of Event
: What was changed, either `X.PropertyNewValue` or `X.PropertyDelete`.

SelectionClear Event
: This event is reported to the owner of a selection when it has gotten a new owner.

window Instance Variable of SelectionClear
: The owner window of the selection.

atom Instance Variable of SelectionClear
: The selection atom.

time Instance Variable of SelectionClear

The server X time when the selection was last changed.

SelectionRequest Event

This event is reported to the owner of a selection when a client requests it by calling convert_selection.

owner Instance Variable of SelectionRequest

The owner window of the selection.

requestor Instance Variable of SelectionRequest
selection Instance Variable of SelectionRequest
target Instance Variable of SelectionRequest
property Instance Variable of SelectionRequest
time Instance Variable of SelectionRequest

These attributes are fetched from the convert_selection call.

SelectionNotify Event

This event is generated by the server if there are no owner of a selection when convert_selection is called. If there is an owner, it should send this event to the requestor when the selection has been converted.

requestor Instance Variable of SelectionNotify
selection Instance Variable of SelectionNotify
target Instance Variable of SelectionNotify
property Instance Variable of SelectionNotify
time Instance Variable of SelectionNotify

See XSelectionEvent(3X11).

ColormapNotify Event

This event is generated when the colormap attribute of a window is set, or when a window's colormap is installed or uninstalled.

window Instance Variable of ColormapNotify

The affected window.

colormap Instance Variable of ColormapNotify

The colormap of the window, or `X.NONE`.

new Instance Variable of ColormapNotify

Set to 1 if the colormap attribute has been set, 0 when the colormap is installed or uninstalled.

state Instance Variable of ColormapNotify

Indicates whether the colormap is installed or not, either `X.ColormapInstalled` or `X.ColormapUninstalled`.

MappingNotify Event

This event is sent to all clients, without any event mask having to be set. It is sent when the keyboard or pointer mapping is changed.

request Instance Variable of MappingNotify

The mapping that has changed, one of `X.MappingModifier`, `X.MappingKeyboard` or `X.Pointer`.

first_keycode Instance Variable of MappingNotify
count Instance Variable of MappingNotify

If the keyboard mapping has changed, this is the range of modified keycodes.

ClientMessage Event

This event is only generated by clients using send_event.

window Instance Variable of ClientMessage

The destination window of the event.

client_type Instance Variable of ClientMessage

The type of message, an atom.

data Instance Variable of ClientMessage

The message data as a tuple: `(format, mdata)`

Format must be one of 8, 16, or 32. mdata must either be a string of exactly 20 characters, or a list of exactly 20, 10 or 5 integers depending of the format.

AnyEvent Event

This event is represents an event the Xlib does not recognise. These should never be returned by `Display.next_event()`, but they might be useful for sending special events to other clients.

detail Instance Variable of AnyEvent

An eight-bit integer.

data Instance Variable of AnyEvent

A string of exactly 28 characters.

5.4 Sending Events

Clients can send events to each others. Most often these are general-purpose `ClientMessage` events, but any event type can be sent.

To send an event, an event object must be created. This is done by instantiating an event class, providing values to all its fields as keyword parameters. The event is then sent with the `send_event()` method:

send_event (*event*, *event_mask = 0*, *propagate = 0*, *onerror = None*) Method on `Window`

Send *event* to this window.

If *event_mask* is 0, the event is sent to the client that created the window. Otherwise the event is sent to every client selecting any of the event types in *event_mask*.

If no clients have selected any of the event types and *propagate* is true, the X server will search for an ancestor of this window on which some client has selected any of the event types. For details, see XSendEvent(3X11).

send_event (*destination*, *event*, *event_mask = 0*, *propagate = 0*, *onerror = None*) Method on `Display`

Send *event* to *destination*, which can either be a `Window` object or a constant: If `X.PointerWindow` is specified, send the event to the window the pointer is in. If `X.InputFocus` is specified and the focus window contains the pointer, send the event to the window that the pointer is in. Otherwise, send the event to the focus window.

As an example, this creates and sends a `ClientMessage` event of the client specific type `HELLO_WORLD` (an atom), to the window `dest` and with the 8-bitformat value `"G'day, mate"` (which must be exactly twenty bytes):

```
cm_event = Xlib.protocol.event.ClientMessage(
    window = dest,
    client_type = HELLO_WORLD,
    data = (8, "G'day mate\0\0\0\0\0\0\0\0\0\0"))

dest.send_event(cm_event)
```

6 X Objects

All X operations are performed by invoking object methods.

The objects and their methods are listed in the following nodes, however with no detailed explanations. The reader are referred to other sources of X documentation for details, e.g. the X protocol specification, the C Xlib documentation or man pages. A vast collection of X documentation links can be found at `http://www.rahul.net/kenton/xsites.html`.

Methods returning a single value will return that directly. Otherwise, an object containing attributes for each of the result values is returned.

Methods returning some value will raise an exception on error. Methods which does not return any value have instead an `on_error` parameter, which can be used to specify an error handler.

6.1 Display

Display information retrieval methods:

get_display_name ( ) Method on `Display`

fileno ( ) Method on `Display`

close ( ) Method on `Display`

set_error_handler (*handler*) Method on `Display`

flush ( ) Method on `Display`

sync ( ) Method on `Display`

next_event ( ) Method on `Display`

pending_events ( ) Method on `Display`

has_extension (*extension*) Method on `Display`

create_resource_object (*type, id*) Method on `Display`

screen (*sno = None*) Method on `Display`

screen_count ( ) Method on `Display`

get_default_screen ( ) Method on `Display`

keycode_to_keysym (*keycode, index*) Method on `Display`

keysym_to_keycode (*keysym*) Method on `Display`

keysym_to_keycodes (*keysym*) Method on `Display`

refresh_keyboard_mapping (*evt*) Method on `Display`

lookup_string (*keysym*) Method on `Display`

rebind_string (*keysym, newstring*) Method on `Display`

X requests methods:

intern_atom (*name, only_if_exists = 0*) Method on `Display`
Returns Card32(atom)

get_atom_name (*atom*) Method on `Display`
Returns String(name)

get_selection_owner (*selection*) Method on `Display`
Returns Window(owner, (X.NONE,))

send_event (*destination, event, event_mask = 0, propagate = 0, onerror = None*) Method on `Display`

ungrab_pointer (*time, onerror = None*) Method on `Display`

change_active_pointer_grab (*event_mask, cursor, time, onerror = None*) Method on `Display`

ungrab_keyboard (*time, onerror = None*) Method on `Display`

allow_events (*mode, time, onerror = None*) Method on `Display`

grab_server (*onerror = None*) Method on `Display`

ungrab_server (*onerror = None*) Method on `Display`

warp_pointer (*x, y, src_window = 0, src_x = 0, src_y = 0, src_width = 0, src_height = 0, onerror = None*) Method on `Display`

set_input_focus (*focus, revert_to, time, onerror = None*) Method on `Display`

get_input_focus ( ) Method on `Display`
Returns Card8('revert_to'), Window('focus', (X.NONE, X.PointerRoot))

query_keymap ( ) Method on `Display`
Returns FixedList('map', 32, Card8Obj)

open_font (*name*) Method on `Display`
Returns Font or None.

list_fonts (*pattern, max_names*) Method on `Display`
Returns List('fonts', Str)

list_fonts_with_info (*pattern, max_names*) Method on `Display`

Returns a list of Object('min_bounds', structs.CharInfo), Object('max_bounds', structs.CharInfo), Card16('min_char_or_byte2'), Card16('max_char_or_byte2'), Card16('default_char'), Card8('draw_direction'), Card8('min_byte1'), Card8('max_byte1'), Card8('all_chars_exist'), Int16('font_ascent'), Int16('font_descent'), Card32('replies_hint'), List('properties', structs.FontProp), String8('name'),

set_font_path (*path, onerror = None*) Method on `Display`

get_font_path ( ) Method on `Display`

Returns List('paths', Str)

query_extension (*name*) Method on `Display`

Returns None, or Card8('major_opcode'), Card8('first_event'), Card8('first_error'),

list_extensions ( ) Method on `Display`

Returns List('names', Str)

change_keyboard_mapping (*first_keycode, keysyms, onerror = None*) Method on `Display`

get_keyboard_mapping (*first_keycode, count*) Method on `Display`

Returns KeyboardMapping('keysyms'),

change_keyboard_control (*onerror = None, **keys*) Method on `Display`

get_keyboard_control ( ) Method on `Display`

bell (*percent = 0, onerror = None*) Method on `Display`

change_pointer_control (*accel = None, threshold = None, onerror = None*) Method on `Display`

accel should be `None` or `(num, denum)`.

get_pointer_control ( ) Method on `Display`

Returns Card16('accel_num'), Card16('accel_denom'), Card16('threshold'),

set_screen_saver (*timeout, interval, prefer_blank, allow_exposures, onerror = None*) Method on `Display`

get_screen_saver ( ) Method on `Display`

Returns Card16('timeout'), Card16('interval'), Card8('prefer_blanking'), Card8('allow_exposures'),

change_hosts (*mode, host_family, host, onerror = None*) Method on `Display`

list_hosts ( ) Method on `Display`

Returns Card8('mode'), List('hosts', structs.Host),

set_access_control (*mode, onerror = None*) Method on `Display`

set_close_down_mode (*mode, onerror = None*) Method on `Display`

force_screen_saver (*mode, onerror = None*) Method on `Display`

set_pointer_mapping (*map*) Method on `Display`
Returns Card8('status'),

get_pointer_mapping ( ) Method on `Display`
Returns List('map', Card8Obj),

set_modifier_mapping (*keycodes*) Method on `Display`
Returns Card8('status'),

get_modifier_mapping ( ) Method on `Display`
Returns ModifierMapping('keycodes')

no_operation (*onerror = None*) Method on `Display`

6.2 Resource

All resource objects can be compared and hashed, so they can be used as mapping indices. They also have this method:

kill_client (*onerror = None*) Method on `Resource`

6.3 Colormap

See Section 6.2 [Resource], page 32, for additional methods on `Colormap` objects.

free (*onerror = None*) Method on `Colormap`

copy_colormap_and_free (*scr_cmap*) Method on `Colormap`
Returns Colormap.

install_colormap (*onerror = None*) Method on `Colormap`

uninstall_colormap (*onerror = None*) Method on `Colormap`

alloc_color (*red, green, blue*) Method on `Colormap`

alloc_named_color (*name*) Method on `Colormap`
Returns None or Card32('pixel'), Card16('exact_red'), Card16('exact_green'), Card16('exact_blue'), Card16('screen_red'), Card16('screen_green'), Card16('screen_blue'),

alloc_color_cells (*contiguous, colors, planes*) Method on `Colormap`
Returns List('pixels', Card32Obj), List('masks', Card32Obj),

alloc_color_planes (*contiguous, colors, red, green, blue*) Method on `Colormap`
Returns Card32('red_mask'), Card32('green_mask'), Card32('blue_mask'), List('pixels', Card32Obj),

free_colors (*pixels, plane_mask, onerror = None*) Method on `Colormap`

store_colors (*items, onerror = None*) Method on `Colormap`

store_named_color (*name, pixel, flags, onerror = None*) Method on `Colormap`

query_colors (*pixels*) Method on `Colormap`
Returns List('colors', structs.RGB),

lookup_color (*name*) Method on `Colormap`
Returns Card16('exact_red'), Card16('exact_green'), Card16('exact_blue'), Card16('screen_red'), Card16('screen_green'), Card16('screen_blue'),

6.4 Cursor

See Section 6.2 [Resource], page 32, for additional methods on `Cursor` objects.

free (*onerror = None*) Method on `Cursor`

recolor ((*fore_red, fore_green, fore_blue*), (*back_red, back_green, back_blue*), *onerror = None*) Method on `Cursor`

6.5 Drawable

`Drawable` is a base class of `Window` and `Pixmap` objects. See Section 6.6 [Window], page 35, and Section 6.7 [Pixmap], page 38.

See Section 6.2 [Resource], page 32, for additional methods.

get_geometry ( ) Method on `Drawable`
Returns Window('root'), Int16('x'), Int16('y'), Card16('width'), Card16('height'), Card16('border_width'),

create_pixmap (*width, height, depth*) Method on `Drawable`
Returns Pixmap.

create_gc (***keys*) Method on `Drawable`
Returns GC.

copy_area (*gc, src_drawable, src_x, src_y, width, height, dst_x, dst_y, onerror = None*) Method on `Drawable`

copy_plane (*gc, src_drawable, src_x, src_y, width, height, dst_x, dst_y, bit_plane, onerror = None*) Method on `Drawable`

poly_point (*gc, coord_mode, points, onerror = None*) Method on `Drawable`

request.PolyPoint(display = self.display, onerror = onerror, coord_mode = coord_mode, drawable = self.id, gc = gc, points = points)

point (*gc, x, y, onerror = None*) Method on `Drawable`

poly_line (*gc, coord_mode, points, onerror = None*) Method on `Drawable`

line (*gc, x1, y1, x2, y2, onerror = None*) Method on `Drawable`

poly_segment (*gc, segments, onerror = None*) Method on `Drawable`

poly_rectangle (*gc, rectangles, onerror = None*) Method on `Drawable`

rectangle (*gc, x, y, width, height, onerror = None*) Method on `Drawable`

poly_arc (*gc, arcs, onerror = None*) Method on `Drawable`

arc (*gc, x, y, width, height, angle1, angle2, onerror = None*) Method on `Drawable`

fill_poly (*gc, shape, coord_mode, points, onerror = None*) Method on `Drawable`

poly_fill_rectangle (*gc, rectangles, onerror = None*) Method on `Drawable`

fill_rectangle (*gc, x, y, width, height, onerror = None*) Method on `Drawable`

poly_fill_arc (*gc, arcs, onerror = None*) Method on `Drawable`

fill_arc (*gc, x, y, width, height, angle1, angle2, onerror = None*) Method on `Drawable`

put_image ( ) Method on `Drawable`

Not implemented yet.

get_image ( ) Method on `Drawable`

Not implemented yet.

draw_text (*gc, x, y, text, onerror = None*) Method on `Drawable`

poly_text (*gc, x, y, items, onerror = None*) Method on `Drawable`

poly_text_16 (*gc, x, y, items, onerror = None*) Method on `Drawable`

image_text (*gc, x, y, string, onerror = None*) Method on `Drawable`

image_text_16 (*gc, x, y, string, onerror = None*) Method on `Drawable`

query_best_size (*item_class, width, height*) Method on `Drawable`

Returns Card16('width'), Card16('height'),

6.6 Window

`Window` objects have additional objects, see Section 6.2 [Resource], page 32 and Section 6.5 [Drawable], page 33.

create_window (*x, y, width, height, border_width, depth, window_class = X.CopyFromParent, visual = X.CopyFromParent, **keys*) — Method on `Window`

Returns Window

change_attributes (*onerror = None, **keys*) — Method on `Window`

get_attributes ( ) — Method on `Window`

Returns Card8('backing_store'), Card32('visual'), Card16('class'), Card8('bit_gravity'), Card8('win_gravity'), Card32('backing_bit_planes'), Card32('backing_pixel'), Card8('save_under'), Card8('map_is_installed'), Card8('map_state'), Card8('override_redirect'), Colormap('colormap', (X.NONE,)), Card32('all_event_masks'), Card32('your_event_mask'), Card16('do_not_propagate_mask'),

destroy (*onerror = None*) — Method on `Window`

destroy_sub_windows (*onerror = None*) — Method on `Window`

change_save_set (*mode, onerror = None*) — Method on `Window`

reparent (*parent, x, y, onerror = None*) — Method on `Window`

map (*onerror = None*) — Method on `Window`

map_sub_windows (*onerror = None*) — Method on `Window`

unmap (*onerror = None*) — Method on `Window`

unmap_sub_windows (*onerror = None*) — Method on `Window`

configure (*onerror = None, **keys*) — Method on `Window`

circulate (*direction, onerror = None*) — Method on `Window`

query_tree ( ) — Method on `Window`

Returns Window('root'), Window('parent', (X.NONE,)), List('children', WindowObj),

change_property (*property, type, format, data, mode = X.PropModeReplace, onerror = None*) — Method on `Window`

delete_property (*property, onerror = None*) — Method on `Window`

get_property (*property, type, offset, length, delete = 0*) Method on `Window`
Returns None or Card32('property_type'), Card8('format'), PropertyData('value'), Card32('bytes_after'),

get_full_property (*property, type, sizehint = 10*) Method on `Window`
Returns None or Card32('property_type'), Card8('format'), PropertyData('value'), Card32('bytes_after'),

list_properties ( ) Method on `Window`
Returns List('atoms', Card32Obj),

set_selection_owner (*selection, time, onerror = None*) Method on `Window`

convert_selection (*selection, target, property, time, onerror = None*) Method on `Window`

send_event (*event, event_mask = 0, propagate = 0, onerror = None*) Method on `Window`

grab_pointer (*owner_events, event_mask, pointer_mode, keyboard_mode, confine_to, cursor, time*) Method on `Window`
Returns Card8('status'),

grab_button (*button, modifiers, owner_events, event_mask, pointer_mode, keyboard_mode, confine_to, cursor, onerror = None*) Method on `Window`

ungrab_button (*button, modifiers, onerror = None*) Method on `Window`

grab_keyboard (*owner_events, pointer_mode, keyboard_mode, time*) Method on `Window`
Returns Card8('status'),

grab_key (*key, modifiers, owner_events, pointer_mode, keyboard_mode, onerror = None*) Method on `Window`

ungrab_key (*key, modifiers, onerror = None*) Method on `Window`

query_pointer ( ) Method on `Window`
Returns Card8('same_screen'), Window('root'), Window('child', (X.NONE,)), Int16('root_x'), Int16('root_y'), Int16('win_x'), Int16('win_y'), Card16('mask'),

get_motion_events (*start, stop*) Method on `Window`
Returns List('events', structs.TimeCoord),

translate_coords (*src_window, src_x, src_y*) Method on `Window`
Returns Window('child', (X.NONE,)), Int16('x'), Int16('y'),

warp_pointer (*x, y, src_window = 0, src_x = 0, src_y = 0, src_width = 0, src_height = 0, onerror = None*) Method on `Window`

set_input_focus (*revert_to, time, onerror = None*) Method on `Window`

clear_area (*x = 0, y = 0, width = 0, height = 0, exposures = 0, onerror = None*) Method on `Window`

create_colormap (*visual, alloc*) Method on `Window`
Returns Colormap.

list_installed_colormaps ( ) Method on `Window`
Returns List('cmaps', ColormapObj),

rotate_properties (*properties, delta, onerror = None*) Method on `Window`

set_wm_name (*name, onerror = None*) Method on `Window`

get_wm_name ( ) Method on `Window`
Returns None or string.

set_wm_icon_name (*name, onerror = None*) Method on `Window`

get_wm_icon_name ( ) Method on `Window`
Returns None or string.

set_wm_class (*inst, cls, onerror = None*) Method on `Window`

get_wm_class ( ) Method on `Window`
Returns None or (isntance, class)

set_wm_transient_for (*window, onerror = None*) Method on `Window`

get_wm_transient_for ( ) Method on `Window`
Returns None or Window.

set_wm_protocols (*protocols, onerror = None*) Method on `Window`

get_wm_protocols ( ) Method on `Window`
Returns list of atoms.

set_wm_colormap_windows (*windows, onerror = None*) Method on `Window`

get_wm_colormap_windows ( ) Method on `Window`
Returns list of Windows.

set_wm_client_machine (*name, onerror = None*) Method on `Window`

get_wm_client_machine ( ) Method on `Window`

Returns None or string.

set_wm_normal_hints (*hints = , onerror = None, **keys*) Method on `Window`

get_wm_normal_hints ( ) Method on `Window`

Returns Card32('flags'), Int32('min_width', default = 0), Int32('min_height', default = 0), Int32('max_width', default = 0), Int32('max_height', default = 0), Int32('width_inc', default = 0), Int32('height_inc', default = 0), Object('min_aspect', Aspect, default = (0, 0)), Object('max_aspect', Aspect, default = (0, 0)), Int32('base_width', default = 0), Int32('base_height', default = 0), Int32('win_gravity', default = 0),

set_wm_hints (*hints = , onerror = None, **keys*) Method on `Window`

get_wm_hints ( ) Method on `Window`

Returns Card32('flags'), Card32('input', default = 0), Set('initial_state', 4, (Xutil.NormalState, Xutil.IconicState), default = Xutil.NormalState), Pixmap('icon_pixmap', default = 0), Window('icon_window', default = 0), Int32('icon_x', default = 0), Int32('icon_y', default = 0), Pixmap('icon_mask', default = 0), Window('window_group', default = 0),

set_wm_state (*hints = , onerror = None, **keys*) Method on `Window`

get_wm_state ( ) Method on `Window`

Returns Set('state', 4, (Xutil.WithdrawnState, Xutil.NormalState, Xutil.IconicState)), Window('icon', (X.NONE,)),

set_wm_icon_size (*hints = , onerror = None, **keys*) Method on `Window`

get_wm_icon_size ( ) Method on `Window`

Returns Card32('min_width'), Card32('min_height'), Card32('max_width'), Card32('max_height'), Card32('width_inc'), Card32('height_inc'),

6.7 Pixmap

`Pixmap` objects have additional objects, see Section 6.2 [Resource], page 32 and Section 6.5 [Drawable], page 33.

free (*onerror = None*) Method on `Pixmap`

create_cursor (*mask, (fore_red, fore_green, fore_blue), (back_red, back_green, back_blue), x, y*) Method on `Pixmap`

Returns Cursor.

6.8 Fontable

`Fontable` is a base class of `GC` and `Font` objects. See Section 6.9 [GC], page 39, and Section 6.10 [Font], page 39.

See Section 6.2 [Resource], page 32, for additional methods.

query ( ) — Method on `Fontable`

Returns Object('min_bounds', structs.CharInfo), Object('max_bounds', structs.CharInfo), Card16('min_char_or_byte2'), Card16('max_char_or_byte2'), Card16('default_char'), Card8('draw_direction'), Card8('min_byte1,'), Card8('max_byte1,'), Card8('all_chars_exist'), Int16('font_ascent'), Int16('font_descent'), List('properties', structs.FontProp), List('char_infos', structs.CharInfo),

query_text_extents (*string*) — Method on `Fontable`

Returns Card8('draw_direction'), Int16('font_ascent'), Int16('font_descent'), Int16('overall_ascent'), Int16('overall_descent'), Int32('overall_width'), Int32('overall_left'), Int32('overall_right'),

6.9 GC

`GC` objects have additional objects, see Section 6.2 [Resource], page 32 and Section 6.8 [Fontable], page 39.

change (*onerror = None, **keys*) — Method on `GC`

copy (*src_gc, mask, onerror = None*) — Method on `GC`

set_dashes (*offset, dashes, onerror = None*) — Method on `GC`

set_clip_rectangles (*x_origin, y_origin, rectangles, ordering, onerror = None*) — Method on `GC`

free (*onerror = None*) — Method on `GC`

6.10 Font

`Font` objects have additional objects, see Section 6.2 [Resource], page 32 and Section 6.8 [Fontable], page 39.

close (*onerror = None*) — Method on `Font`

create_glyph_cursor (*mask, source_char, mask_char, (fore_red, fore_green, fore_blue), (back_red, back_green, back_blue)*) — Method on `Font`

Returns Cursor.