Xlib - C Language X Interface
Protocol Version 11

Jim Gettys

Digital Equipment Corporation
Systems Research Center
MIT Project Athena

Ron Newman

Massachusetts Institute of Technology
MIT Project Athena

Robert W. Scheifler

Massachusetts Institute of Technology
Laboratory for Computer Science

ABSTRACT

The X Window System± is a network transparent windowing system developed at MIT which runs under 4.3BSD UNIX† and Ultrix-32‡, many other UNIX variants, VAX/VMS as well as several other systems.

X display servers run on computers with either monochrome or color bit-map terminals. The server distributes user input to, and accepts output requests from various client programs located either on the same machine or elsewhere in your network. Xlib is a C subroutine library that application programs ('clients') use to interface with the window system via a stream connection. While a client normally runs on the same machine as the X server it is talking to, this need not be the case.

This document is a descriptive guide to the low-level C language interface to the X Window System protocol. It is neither a tutorial or a user guide to programming to the X Window System. Other high-level abstraction (for example, those provided by the toolkits for X) are built on top of the Xlib library. For further information about these high-level libraries, see the appropriate toolkit documentation. For general information about the design of X, see ''The X Window System.''

June 19, 1987

Xlib - C Language X Interface
Protocol Version 11

Jim Gettys

Digital Equipment Corporation
Systems Research Center
MIT Project Athena

Ron Newman

Massachusetts Institute of Technology
MIT Project Athena

Robert W. Scheifler

Massachusetts Institute of Technology
Laboratory for Computer Science

1. Introduction to Xlib Table of Contents Introduction

To program using Xlib, you should have a basic understanding of a graphics window system. To provide an introduction to Xlib programming, this chapter discusses:

- Overview to the X Window System
- Naming and argument conventions
- Programming considerations

1.1. Overview to X Window System

The X window system supports one or more screens containing overlapping (sub)windows. A screen is a physical monitor and hardware, which may be either color or black and white. There can be multiple screens per display or workstation. All the windows in an X server are arranged in a strict hierarchy. At the top of the hierarchy are the 'root' windows, which cover each of the display screens. Each root window is partially or completely covered by child windows. All windows have parents except for the root windows. There is usually at least one window per application program. Child windows may in turn have their own children. In this way, an application program can create a tree of arbitrary depth on each screen.

A child window may be larger than its parent--that is, part or all of the child window may extend beyond the boundaries of the parent. However, all output to a window is clipped by the

---

† UNIX is a trademark of AT&T Bell Laboratories.

‡ Ultrix, Ultrix-32, Ultrix-32m, and Ultrix-32w, VAX/VMS are trademarks of Digital Equipment Corporation.

± The X Window System is a trademark of MIT.

boundaries of its parent window.

If several children of a window have overlapping locations, then one of the children is considered to be 'on top of' or 'raised over' the others, obscuring them. Output to areas covered by other windows will be suppressed by the window system. If a window is obscured by a second window, then the window will obscure only those ancestors of the second window which are also ancestors of the window.

A window has a border of zero or more pixels in width, which can be any pattern (pixmap) or solid color you like. A window usually (but not always) has a background pattern which will be repainted by the window system when uncovered. Each window has its own coordinate system. Normally, child windows obscure their parents unless no background has been defined, and graphic operations in the parent window usually are clipped by the children.

Input from X takes the form of 'events'. Events may either be side effects of a command (for example, restacking windows generates exposure events), or completely asynchronous (for example, the keyboard). A client program asks to be informed of events; X never sends events a program did not ask for.

X does not take responsibility for the contents of windows; when (part or all of) a window is hidden and then brought back onto the screen, its contents may be lost and the client program is notified (by an exposure event) that (part or all of) the window needs to be repainted. Programs must be prepared to regenerate the contents of windows on demand.

Most of the subroutines in Xlib just add requests to an output buffer; these requests later execute asynchronously on the X server (often referred to as ''display server'' below). Subroutines that return values do not return (''block'') until an explicit reply is received or an error occurs. This may be queried information or a ''resource id'' discussed below. If such a call results in an error, the error will generally not be reported (by a call to an optional error handler) until some later, blocking call is made.

If a client does not want a request to execute asynchronously, he can follow it with a call to XSync, which will block until all previously buffered asynchronous events have been sent and acted on.

As an important side effect, the output buffer is always flushed by a call to any subroutine which returns a value or waits for input (for example, XPending, XNextEvent, XWindowEvent, XFlush, or XSync).

Many subroutines will return an integer resource id. These can be of type Window, Font, Pixmap, Bitmap, Cursor, and GContext, as defined in <X11/X.h>. Some subroutines return ''Status'', an integer error indication. If the subroutine fails, it will return 0.

Since C does not provide multiple return values, many subroutines must return their results by writing into client-passed storage. Any pointer that is used to return a value is designated as ''/* RETURN */'' in the procedure declarations below. All other pointers passed to these subroutines are used for reading only. If such a procedure returns a status of 0, then it has NOT updated the return parameters. By default, errors are handled by a standard library subroutine, or you can provide your own. Subroutines that return pointers to strings will return NULL pointers if the string does not exist.

Input events (e.g. key pressed, mouse moved) arrive asynchronously from the server and are queued until they are requested by a call to XNextEvent or XWindowEvent. In addition, some of the library subroutines (for example, XResizeWindow and XRaiseWindow) generate ''exposure'' events--requests to repaint sections of a window that do not have valid contents. These events also arrive asynchronously, but the client may wish to explicitly wait for them by calling XSync after calling a subroutine which may generate exposure events.

1.2. Naming and Argument Conventions within Xlib

Throughout the Xlib library, a number of conventions for naming and syntax of the Xlib functions have been followed. These conventions, are intended to make the syntax of the functions more predictable, given you remember what information the routine may require.

The major conventions are:

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

1.3. Programming Considerations

*** TBS ***

1.4. Conventions Used in this Manual

*** TBS ***

2. Display Functions Initializing the Window System

Xlib provides functions that allow you to specify which displays you want to perform window operations on. Xlib also provides macros that return information about a specific display. This chapter discusses how to:

- Open the display
- Obtain information about a display
- Generate a NoOperation protocol request
- Close (disconnect) a display

In addition, the chapter concludes with a section that describes the operations that occur when the connection to the X server is closed.

2.1. Opening the Display

To open or connect a display to an X server, use XOpenDisplay. The definition for this function is:

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

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

The display name or DISPLAY environment variable is a string that has the format:

hostname:*number* or *hostname*:*number*.*screen*

hostname Specifies the name of the host machine on which the display is physically connected. You follow the hostname with either a single colon (:) or a double colon (::). Each of these situations is discussed in the following paragraph.

number Specifies the number of the display on that host machine. You optionally follow this display number with a period (.).

screen Specifies the number of the screen on that host machine. Multiple screens can be connected to (controlled by) a single X server, while only one display can be connected to (controlled by) a single X server. The screen sets an internal variable that can be accessed by using the DefaultScreen macro. For further information, see the section on Obtaining Information About a Display. If the screen is omitted, it defaults to 0.

For example, the following would specify screen 2 of display 0 on the machine named mit-athena:

mit-athena:0.2

The XOpenDisplay routine connects the specified hardware display to the server through TCP, Unix, or DECnet streams. If the hostname is a host machine name and a single colon (:) separates the hostname and display number, XOpenDisplay connects the hardware display to TCP streams. If the hostname is unix and a single colon (:) separates it from the display number, XOpenDisplay connects the hardware display to Unix domain IPC streams. If the hostname is a host machine name and a double colon (::) separates the hostname and display number, XOpenDisplay connects the hardware display to DECnet streams. To use DECnet, however, you must build all software for DECnet. A single X server will accept both TCP and DECnet connections if it has been built for DECnet.

If successful, XOpenDisplay returns a pointer to a Display structure that is defined in <X11/Xlib.h>. For information about using macros to obtain information from the Display structure, see the following section. If XOpenDisplay does not succeed, it returns a NULL.

After a successful call to XOpenDisplay, all of the screens in the display may be used by the application. The screen number specified in the display_name argument serves only to specify the value that will be returned by the DefaultScreen macro.

2.2. Obtaining Information About a Display

Once you have successfully connected (opened) your display to the X server, you can obtain data from the Display structure associated with that display. The Xlib library provides a number of useful macros and corresponding functions which return data from the Display structure. The macros are to be used for C programming, while their corresponding functions are for other language bindings.

2.2.1. Information Macros

The C language macros, arranged in alphabetical order, are:

AllPlanes	Returns a value with all bits set on suitable for use in a plane argument to a procedure.
BitmapBitOrder(*display*)	Within each bitmap unit, the leftmost bit in the bitmap as displayed on the screen is either the least or most significant bit in the unit. This macro can return one of these constants: LSBFirst or MSBFirst.
BitmapPad(*display*)	Each scanline must be padded to a multiple of bits specified by the value returned by this macro.
BitmapUnit(*display*)	Returns the size of a bitmap's unit. The scanline is quantized (calculated) in multiples of this value.
BlackPixel(*display*, *screen*)	Returns the black pixel value for the specified screen.
ConnectionNumber(*display*)	Returns the file descriptor of the connected display. On a UNIX-based system, you can then pass this returned file descriptor to the select system call when your application program is driving more than one display at a time.
DefaultColormap(*display*, *screen*)	Returns the default Colormap ID for allocation on the specified screen. Most routine allocations of color should be made out of this color map.
DefaultDepth(*display*, *screen*)	Returns the depth (number of planes) of the default root window for the specified screen. Other depths may also be supported on this screen. See XMatchVisualInfo in the utilities chapter to find out how to determine what depths may be available.
DefaultGC(*display*, *screen*)	Returns the default graphics context for the root window of the specified screen. This GC is created for the convenience of simple applications and contains the default GC components with the foreground and background pixel values initialized to the white and black pixels, respectively, for the screen.

DefaultScreen(*display*)	Returns the default screen referenced in the XOpenDisplay routine. This macro should be used to retrieve the screen number in applications that will only use a single screen.
DefaultVisual(*display*, *screen*)	Returns the default visual type for the specified screen.
DisplayCells(*display*, *screen*)	Returns the number of entries in the default color map.
DisplayHeight(*display*, *screen*)	Returns an integer that describes the height of the screen in pixels.
DisplayHeightMM(*display*, *screen*)	Returns the height of the specified screen in millimeters.
DisplayPlanes(*display*, *screen*)	Returns the depth of the root window.
DisplayString(*display*)	Returns the string that was passed to XOpenDisplay when the current display was opened. On UNIX-based systems, if the passed string was NULL, this macro returns the value of the DISPLAY environment variable when the current display was opened. This macro is useful to applications which invoke the fork system call and want to open a new connection to the same display from the child process.
DisplayWidth(*display*, *screen*)	Returns an integer that describes the width of the screen in pixels.
DisplayWidthMM(*display*, *screen*)	Returns the width of the specified screen in millimeters.
ImageByteOrder(*display*)	Specifies the required byte order for images for each scanline unit in XYFormat (bitmap) or for each pixel value in ZFormat. The macro can return one of these constants: LSBFirst or MSBFirst.
ProtocolRevision(*display*)	Returns the minor protocol revision number of the X server.
ProtocolVersion(*display*)	Returns the version number of the X protocol associated with the connected display.
QLength(*display*)	Returns the length of the input queue for the connected display.
RootWindow(*display*, *screen*)	Returns the root window. This macro is useful with subroutines that take a parent window as an argument.
ScreenCount(*display*)	Returns the number of available screens.

ServerVendor(*display*)	Returns pointer to a null terminated string giving some identification of the owner of the X server implementation.
VendorRelease(*display*)	Returns a number related to a vendor's release of the X server.
WhitePixel(*display*, *screen*)	Returns the white pixel value for the specified screen.

All other members of the Display structure are private to Xlib and must not be used. That is, applications must never directly modify elements in the Display structure.

2.2.2. Information Functions

The following Xlib functions, listed in alphabetical order, are to be used with other language bindings in place of the C language macros.

unsigned long XAllPlanes()

XAllPlanes returns a value with all bits set on suitable for use in a plane argument to a procedure.

int XBitmapBitOrder(*display*)
    Display **display*;

Within each bitmap unit, the leftmost bit in the bitmap as displayed on the screen is either the least or most significant bit in the unit. This function can return one of these constants: LSBFirst or MSBFirst.

int XBitmapPad(*display*)
    Display **display*;

Each scanline must be padded to a multiple of bits specified by the value returned by this function.

int XBitmapUnit(*display*)
    Display **display*;

XBitmapUnit returns the size of a bitmap's unit. The scanline is quantized (calculated) in multiples of this value.

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

XBlackPixel returns the black pixel value for the specified screen.

int XConnectionNumber(*display*)
    Display **display*;

XConnectionNumber returns the file descriptor of the connected display. You can then pass this returned file descriptor to the select system call when your application program is driving more than one display at a time.

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

XDefaultColormap returns the default Colormap ID for allocation on the specified screen. Most routine allocations of color should be made out of this color map.