

The X Window System

The **X Window System** is a networking and display protocol which provides windowing on bitmapped displays. X provides the basic framework for building GUI environments, such as drawing and moving windows on the screen and interacting with a mouse and/or a keyboard. The X project was started at MIT in 1984. The current release protocol, X11, was first released in 1987.

X is based on a client-server model of computing. An *X server* program runs on a computer with a graphical display and communicates with various *client programs*. The server accepts requests for graphical output (windows) and sends back user input (keyboard, mouse).

In X, the server runs on the user's computer, while the clients may run on a different machine. This is the reverse of the common configuration of client-server systems, where the *client* runs on the user's computer and the server runs on a remote computer. This reversal often confuses new X users. The X Window terminology takes the perspective of the program, rather than the end-user or the hardware: the remote programs connect to the X server display running on the local machine, and thus act as clients; the local X display accepts incoming traffic, and thus acts as a server.

Why X?

X was designed to be portable but fast: it had to run on many operating systems and many different hardware configurations and yet have good performance.

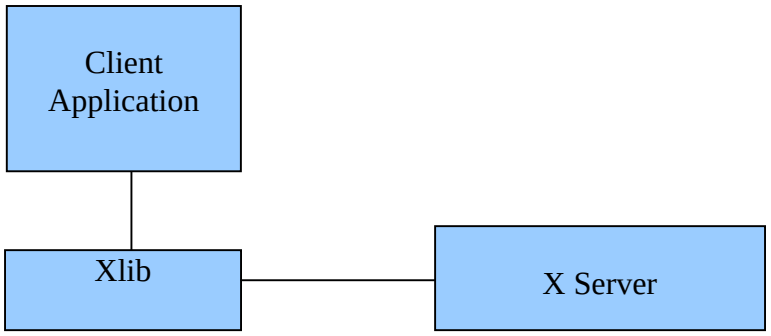
X was designed to work across networks. In other words, it was designed so that the client application could run on one machine and the server on another machine in the same network. It was designed to work across networks regardless of the underlying network protocols. This was achieved by writing a low level protocol that X itself would use.

X does not require a particular style of user interface (unlike Microsoft Windows and Apple's OS, which both had user interface guidelines for developers.) It can work with any style of GUI. X was essentially “policy free”, very much like the way UNIX itself was designed.

X is widely used on UNIX systems and there are ports to Windows, Apple OS, and other major vendors' systems.

The Xlib Library

Programs that are written to interact with the X Windows System use the Xlib library, which is simply the collection of primitives and data types that act as an interface to X. Xlib was originally written as a C library, but there are ports of it for other languages as well.



Events in X Windows

Events are packets sent by the server to the client to communicate that something the client may be interested in has happened. A client can request the server to send an event to another client; this is used for communication between clients. For example, when a client requests the text that is currently selected in a text box, an event is sent to the client that is currently handling the window that holds the selection.

Another example is the `Expose` event. The content of a window may be destroyed in certain circumstances, such as when a window is covered by another window. When the obscured portion of the window is made visible, such as by making the window the foreground window, the X server generates an `Expose` event to notify the client that part of the window has to be redrawn.

Other examples of events are those that notify clients of keyboard or mouse input, of window resizing or moving, and so on.

Some kinds of events are always sent to client, but most kinds of event are sent only if the client previously indicated to the server that it wanted to know about them. For example, a client program may choose to ignore keyboard events and use only mouse clicks, such as a clock display, or it may do the reverse; it may ignore the mouse and handle only keyboard events.

Well then, what is an event?

An event is a packet of information that is generated by the server when certain actions occur. The packet is queued by the server for the client to access when it is ready. Even though the events are placed in a queue, they are not necessarily processed in FIFO order; the client may choose to process events that arrived later than others. Usually, though, they are read and processed in the order in which they occurred.

Some specific events are:

- Mouse (or other pointer) button pressed or released. (**ButtonPress**, **ButtonRelease**)
- Window mapped or unmapped. (**MapNotify**, **UnmapNotify**)
- Mouse crossing a window boundary. (**EnterNotify**, **LeaveNotify**)

These event types are usually used for user input and to control a user interface.

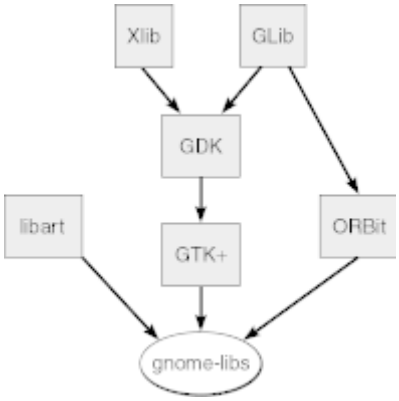


Illustration 1: The GNOME

Another group of events reports side effects of window operations such as the **Expose** event mentioned above.

A third group of events allows various clients to communicate with each other and with the window manager. For example:

- A client may request that all keyboard input be sent to a particular window regardless of the pointer position; this is called a *keyboard focus* window. Changing keyboard focus from one window to another causes **FocusIn** and **FocusOut** events, indicating to the client whether or not it can expect further keyboard events.
- Changing the mapping between keyboard keys and codes they generate causes a **MappingNotify** event to be sent to all clients.
- A **PropertyNotify** event is generated when a client changes a property on a window.
- **SelectionClear**, **SelectionNotify**, and **SelectionRequest** events are used to communicate back and forth between a client that is allowing a user to select a section of text (or other information) and a client that is allowing the user to place the information in its window. Some of these events are sent with **XSendEvent**.

GIMP and GTK+

GIMP originally stood for ***G**eneral **I**mage **M**anipulation **P**rogram*. Its creators, Spencer Kimball and Petter Mattis, initially started GIMP as a semester-long project for a class at the University of California in Berkeley. Both were integral members of experimental Computing Facility, a student club at Berkeley. In 1997, after both Kimball and Mattis had graduated from Berkeley, the name was changed to the ***G**NU **I**mage **M**anipulation **P**rogram* when it became an official part of the GNU Project. A version 2 was released in 2005. [1]

GIMP is a freely distributed piece of software for such tasks as photo retouching, image composition and image authoring. It works on many operating systems, in many languages. GIMP's user interface is built using GTK+, the GIMP ToolKit. The GTK+ library was initially a

1 The information in this section was extracted from an article in Wikipedia.

part of the GIMP source tree. It subsequently took on a life of its own. It is now in stable version 2.12.

GLIB

GLib is a cross-platform, general-purpose, software utility library that provides many data types and macros. It was developed as part of the GTK+ project, but is now used by other applications. While it was originally a convenient library to collect low-level code in, it has since expanded into offering wrapper functions for functionality that is typically different across platforms, including Win32 and Mac OS X.

1. Glib contains, for example, definitions of various data structures such as singly- and doubly-linked lists, regular expressions, timers, threads, memory allocation, and I/O channels. It is, in short, a library with many useful types, macros, and functions. The present version of GTK+ depends upon the GLib library.

GDK

GDK is an acronym for the **G**IMP **D**rawing **K**it. It is a graphics library that provides primitives for raster graphics (e.g., bitmaps), color rendering, line drawing, font and cursor handling, window events, and drag-and-drop functionality. Originally, GDK was developed as an interface to the X server to make it easier to program using the X server. The data types and functions in the GDK are for the most part just wrappers for X data types and functions respectively. For example, X has a data structure called a Pixmap and GDK has a corresponding GdkPixmap.

GDK has now been ported so that it can communicate with the Win32 graphics API and Quartz, the Mac OS X graphics API.

GTK+

GTK+ is a powerful, cross-platform library of tools for creating the visual building blocks of applications. It uses the GDK library to allow your application to create windows, graphic controls such as buttons and drop-down lists, color bitmaps, and more. It also takes advantage of Glib's powerful primitives. Because GTK+ is built on GDK and GLib, it is platform-independent. For simplicity, henceforth, I will use GTK as shorthand for GTK+.

GTK Building Blocks

In GTK, everything is viewed as an object., known as a `GtkObject` in the library. The single most important `GtkObject` is the `GtkWidget.` The word “widget” comes from the word “gadget”; in computer science, the word has come to mean something that responds to mouse clicks, key presses, or other types of user actions. In short, widgets are things like windows, buttons, menus, and edit-boxes.

Widgets

Although GTK is written in C and provides an interface in C, it is object-oriented. It uses clever techniques to provide all of the important features of object-oriented programming, including

private data, encapsulation, and virtual function calls. The top of the object hierarchy is the `GtkObject`. The `GtkWidget` derives from `GtkObject` and inherits all of its properties and methods. The `GtkWindow` is a widget that derives from `GtkWidget` and the `GtkDialog` is a kind of window that derives from `GtkWindow`.

You may think that all widgets are either containers like windows and boxes of various kinds, or controls like buttons and menus, but this is not the case. In fact, some widgets are neither, like a `GtkRange`, which is a range of values. There are also objects that are not widgets, such as the `GtkCellRenderer` class.

Widgets can be grouped together into composite widgets. This is how menus and menu bars are constructed. A menu is a collection of widgets that act together. Menu bars are collections of menus.

Widgets have properties, such as their state and visibility. Some properties are inherited from the ancestral `GtkWidget` class because they are part of all widgets. Others are specific to particular kinds of widgets. Windows, for example, have properties not shared by rulers or calendars. Widgets also have methods, which are the functions that act upon them, and these too are inherited from ancestor classes or are specific to the class itself.

Object Types and Classes

C does not have true classes or inheritance and yet GTK+ has created a class hierarchy. The programming is extremely clever and complex. While I do not intend to cover it here, the fact of it has consequences for how programs are written using GTK. GTK's runtime system registers types dynamically as they are instantiated and used, and the library provides a set of macros that enable the runtime type-checking to take place. The following code shows an example of the kind of runtime type checking that is used.

```
#include <gtk/gtk.h>
int main (int argc, char *argv[])
{
    gtk_init (&argc, &argv);
    GtkWindow *window = GTK_WINDOW (gtk_window_new(GTK_WINDOW_TOPLEVEL));
    GtkButton *button = GTK_BUTTON (gtk_button_new(  ));

    if(GTK_IS_WINDOW(window))
      g_message("TRUE: window is a window");
    if(GTK_IS_CONTAINER(window))
      g_message("TRUE: window is a container");
    if(GTK_IS_CONTAINER(button))
      g_message("TRUE: button is a container");
    if(!GTK_IS_WINDOW(button))
      g_warning("FALSE: button is not derived from window");
    return 0;
}
```

Notice the casting macros and the type checking macros in the code above. This is typical of the way GTK programs look. Notice that the `GTK_IS_CONTAINER` macro is applied to a `GtkButton*` object. The actual type of the pointer is being tested dynamically at runtime, and this is C, not C++. It takes some doing to make this happen.

Signals and Events

All GUIs depend upon the propagation of events from the hardware and/or operating system to the application, so that it can handle those events. Events usually represent inputs from a user, such as mouse clicks, mouse dragging, or key presses, but they can also represent changes of focus caused by an application becoming active, or remapping of screen regions to different applications. GTK distinguishes between events and signals. Events are derived from the underlying GDK event system, which in turns comes from X Windows. Signals were added to GTK because events were not robust and flexible enough. It is summarized well by Havoc Pennington[2]:

"Events are a stream of messages received from the X server. They drive the Gtk main loop; which more or less amounts to "wait for events, process them" (not exactly, it is really more general than that and can wait on many different input streams at once). Events are a Gdk/Xlib concept."

"Signals are a feature of GtkObject and its subclasses. They have nothing to do with any input stream; really a signal is just a way to keep a list of callbacks around and invoke them ("emit" the signal). There are lots of details and extra features of course. Signals are emitted by object instances, and are entirely unrelated to the Gtk main loop. Conventionally, signals are emitted "when something changes" about the object emitting the signal."

"Signals and events only come together because GtkWidget happens to emit signals when it gets events. This is purely a convenience, so you can connect callbacks to be invoked when a particular widget receives a particular event. There is nothing about this that makes signals and events inherently related concepts, any more than emitting a signal when you click a button makes button clicking and signals related concepts."

2 Havoc Pennington, "GTK+/Gnome Application Development, New Riders Publishing, ISBN 0-7357-0078-8