Lesson 1: The Foundation of GTK+

Versions of GTK+

There are presently three versions of GTK+: 1, 2, and 3. They are mutually incompatible in the sense that programs written and built using any one of them may not run on a system that has only the runtime libraries of a different version. GTK+3 was released in 2011 and represents a major departure from GTK+2, which was at the time a major departure from GTK+1. Unless stated otherwise, these notes describe GTK+ 2. For simplicity, the notes may sometimes refer to "GTK" as a shorthand for GTK+.

The Origins of GTK+

In 1995, Spencer Kimball and Petter Mattis developed an image manipulation program that they named the *General Image Manipulation Program*, or *GIMP* for short, for a class at the University of California in Berkeley. They were integral members of the experimental Computing Facility, a student club at Berkeley. They released GIMP for public use in 1996. In 1997, after both Kimball and Mattis had graduated from UC Berkeley, it became part of the GNU Project and its name was changed to the GNU Image Manipulation Program. A second version 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 was originally built using Motif, but Peter Mattis was dissatisfied with it and developed his own GUI toolkit to rewrite GIMP. He named this the *GIMP Tool Kit*, or *GTK* for short. It went through a sequence of revisions and major changes, leading from GTK 1 to GTK+2, and then to GTK+ 3.

The GTK+ library was initially a part of the GIMP source tree. It subsequently took on a life of its own. GTK+ is a powerful, cross-platform library of tools for creating the visual building blocks of applications. It uses the GDK library to allow applications 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 top of GDK and GLib, it is platform-independent.

GTK+ is one of several libraries upon which the Gnome 2 desktop environment is built. Gnome is the GNU Project's desktop environment for Linux systems. Figure 1 shows how various libraries are used by Gnome 2. Libart is a vector-based 2D library with antialiasing and alpha composition that served as a foundation layer for the development of GUI-based applications in Gnome. It has largely been replaced by Cairo (described below).

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).

[1] Courtesy of Wikipedia

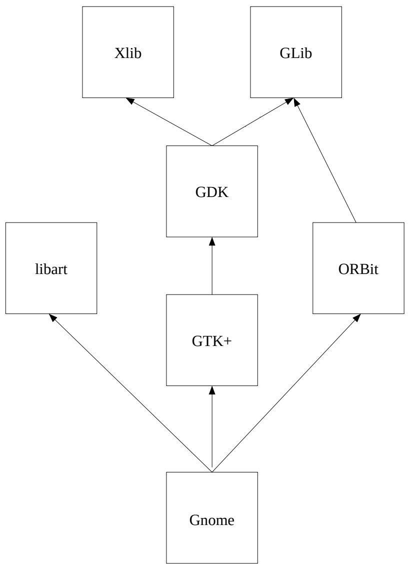

Figure 1: The Gnome 2 dependency graph

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. Figure 2 depicts the relationships between the client program, the Xlib library and the X server executable. Xlib was originally written as a C library, but there are ports of it for other languages as well.

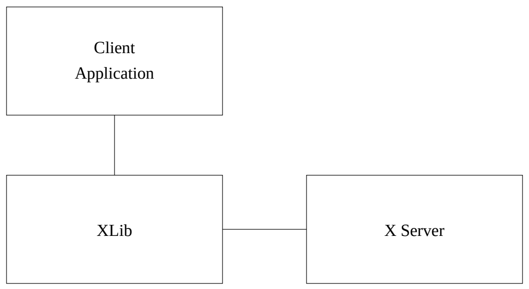

Figure 2: The role of Xlib

Events in X Windows

An *event* is a packet of information that is generated by the server for a client 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.

A client can also 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 second window covers it. When the obscured portion of the first window is made visible, such as by bringing the window to the foreground, 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 the client, but most kinds of events are sent only if the client previously indicated to the server that it wanted to know about them. For example, a client program, such as a clock display, may choose to ignore keyboard events and use only mouse clicks; another program may do the reverse – it may ignore the mouse and handle only keyboard events.

Some specific event types 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. 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 *MapNotify* 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*.

GLib

GLib is a cross-platform, general-purpose, software utility library that provides many data types and macros. It is used for non-graphical purposes. 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.
2. GLib contains hashes, file manipulation, internationalization support, warnings, debugging flags, dynamic module loading, and automatic string completion.
3. GLib provides support for memory management, including slab allocation and memory slices.

GObject

GObject implements a fully featured object-oriented interface in C. This system is the base for the GTK+ widget hierarchical structure as well as for many of the objects implemented in GTK+'s supporting libraries. GObject's object-oriented interface is implemented in part by a generic, dynamic type system called *GType*. GType allows programmers to implement many different dynamic data types through a singly-inherited class structure. Along with the ability to create extensible data types, GObject provides programmers with many fundamental data types.

GDK

GDK is the acronym for the *GIMP Drawing Kit*. 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 were for the most part just wrappers for X data types and functions respectively found in XLib. 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. This is what makes it possible to write a GTK+ application that can run on a wide range of operating systems with different windowing systems.

Gdk-Pixbuf

GdkPixbuf is an application programming interface (*API*) for client-side image manipulation. It was once a separate library but it is now a part of GDK. It provides functions for image loading and pixel buffer manipulation. It also provides functions for progressive image loading, animation, and rendering to things known as *drawables* in GTK+. One of the major advantages of using GdkPixbufs for image manipulation is that they are reference-counted[2]. This concept will be explained in depth later. What it means, in a nutshell, is that an image can be shared by multiple widgets or displays and that the library will know exactly when to free the memory when it is no longer needed by the application.

Client-side image manipulation can sometimes be an advantage. Consider the situation in which an application with a graphical user interface is started up on a remote machine, but its graphical interface is displayed on the local machine. Remember that the client side is where the application is, i.e., the remote machine. The server side is where the underlying windowing system is (the X server, for example), which is the local machine. Suppose that the application needs to manipulate individual pixels of the image, perhaps inverting colors or applying various filters to the image. If the image were stored on the server side, then each of those pixels would have to be sent across the network to the application, which would make its changes and then send them back to the server, again across the network. It would be painfully slow. In contrast, when the image is on the client side, the application can manipulate those pixels quickly and the image is transferred across the network when it is ready to be redisplayed.

Pango

Pango is a cross-platform, cross-toolkit, low-level library for layout and rendering of text, with an emphasis on internationalization. Pango can be used anywhere that text layout is needed, although it was initially created to support GTK+. Presently, Pango forms the core of text and font handling for GTK+-2.x. The name Pango is a portmanteau – "pan" from the Greek for "all", and "go" from the Japanese for "language."

Pango is designed to be modular; the core Pango layout engine can be used with different font back-ends. There are presently three basic back-ends, for UNIX systems, Windows, and Mac OS. On Linux, Pango uses the FreeType and fontconfig libraries for client-side fonts.

Pango supports a vast array of languages. Almost every major script is supported. All text within Pango is represented internally with UTF-8 encoding. UTF-8 is used because it is compatible with 8-bit software, which is prevalent on UNIX platforms. Offsets in UTF-8 are calculated based on characters, not bits, because characters can be multiple bytes. Pango supports a wide variety of text attributes, including language, font family, style, weight, stretch, size, foreground color, background color, underline, strike-through, rise, shape, and scale.

Cairo

Cairo is a two-dimensional, vector-based graphics library with support for multiple output devices, including the X Window System, Quartz, and Win32. In 2005, GTK+ started using Cairo for rendering its widgets. GDK provides access to various Cairo primitives, making it possible to create Cairo drawings directly using the GDK library.

Language Bindings

For those who do not know the term “language binding”, this refers to an interface in the given language to the given software entity. For example, saying that GTK+ has a C++ language binding means that there is a C++ library that gives access to all of the GTK+ interfaces. The number of language bindings for various releases of GTK+2 is extensive. Some of them are:

- C++: Gtkmm (including GLibmm, Libglademm, etc.).

[2]This is a lot like hard links to files in UNIX.

- Python: PyGTK
- Perl: Gtk2-perl
- PHP: PHP-GTK
- Java: Java-Gnome
- C#: Gtk#
- Haskell