GTK+ Programming Notes & Code

Yash Goswami

Refernces : Foundations of GTK+ Developement, Book

A Brief History of GTK+

1. The GIMP Toolkit (GTK+) was originally designed for a raster graphics editor called the GNU Image Manipulation Program (GIMP).

2. Three individuals, Peter Mattis, Spencer Kimball, and Josh MacDonald created GTK+ in 1997 while working in the eXperimental Computing Facility at the University of California, Berkeley.

3. Licensed under the Lesser General Public License (LGPL), GTK+ was adopted as the default graphical toolkit of GNOME and XFCE, two of the most popular Linux desktop environments.

4. While it was originally used on the Linux operating system, GTK+ has since been expanded to support other UNIX-like operating systems: Microsoft Windows, BeOS, Solaris, Mac OS X, and others.

GTK+ and Supporting Libraries

1. GTK+ relies on multiple libraries, each providing the graphical application developer a specific class of functionality.

2. GTK+ is an object-oriented application programming interface (API) written in the C programming language.

3. It is implemented with the concept of classes in mind to create an extensible system that builds upon itself.

4. The object-oriented framework used was originally developed as a part of the GTK+ library itself, but has since been split from GTK+ and added to GLib as a separate supporting library called GObject. GObject enables full object-orientated development in C, including object inheritance, polymorphism, and, to the extent permissible in C, data hiding.

GLib

1. GLib is a general-purpose utility library that is used to implement many useful nongraphical features.

2. While it is required by GTK+, it can also be used independently. Because of this, some applications use GLib without the other GTK+ libraries for the many capabilities it provides.

3. Provides a cross-platform interface.

4. Provides vast array of data types to the developers.
5. Provides a number of data types to C programmers that are usually included by default in other

languages, such as singly and doubly linked lists. Other basic data types include double-ended queues, self-balancing binary trees, and unbalanced n-ary trees.

6. File management, pipes, threads, and more.

Gobject

1. The GLib Object System (GObject) was originally a part of the GTK+ 1 library in the form of the GtkObject class. With the release of GTK+ 2.0, it was moved into its own library, distributed along with Glib.

2. GObject provides GTK+ with two other vital data types: GValue and GObject.

3. GValue is a generic container that can hold any structure of which the system is already aware. This allows functions to return a piece of data of an arbitrary type. Without GValue, the object-oriented nature of GTK+ would not be possible.

4. G_TYPE_GOBJECT, or GObject, is the fundamental type that the widget class inheritance structure of GTK+ is based on. It allows widgets to inherit the properties of their parents, including style properties and signals.

5. GObject also provides widgets with a signal system, an object properties system, and memory management.

GDK

1. The GIMP Drawing Kit (GDK) is a computer graphics library originally designed for the X Window System that wraps around low-level drawing and window functions.

2. GDK acts as the intermediary between Xlib and GTK+. It renders drawings, raster graphics, cursors, and fonts in all GTK+ applications.

3. Also, since it is implemented in every GTK+ program, GDK provides drag-and-drop support and window events. GDK provides GTK+ widgets the ability to be drawn to the screen.

Pango

1. While GDK handles rendering images and windows, Pango controls text and font output in conjunction with Cairo or Xft, depending on your GTK+ version.

2. It can also render directly to an in-memory buffer without the use of any secondary libraries.

ATK

The Accessibility Toolkit (ATK) provides all GTK+ wid- gets with a built-in method of handling accessibility issues. Some examples of things ATK adds support for are screen readers.

First GTK+ Program

```
/* sample.c */

#include <gtk/gtk.h>

int main (int argc, char *argv[])

{
  GtkWidget *window;

  /* Initialize GTK+ and all of its supporting libraries. */
  gtk_init (&argc, &argv);

  /* Create a new window, give it a title and display it to the user. */
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  gtk_window_set_title (GTK_WINDOW (window), "Hello World");
  gtk_widget_show (window);

  /* Hand control over to the main loop. */
  gtk_main ();
  return 0;

}
```

1. Compiling it :
   gcc -o output sample.c `pkg-config –cflags –libs gtk+-2.0`

2. The <gtk/gtk.h> file includes all of the widgets, variables, functions, and structures available in GTK+ as well as header files from other libraries that GTK+ depends on, such as <glib/glib.h> and <gdk/gdk.h>.

3. It produces a top-level GtkWindow widget with a default width and height of 200 pixels. There is no way of exiting the application except to kill it in the terminal where it was launched.

GTK+ Initialization

1. By calling gtk_init(), all initialization work is automatically performed for you.

2. It begins by setting up the GTK+ environment, including obtaining the GDK display and preparing the GLib main event loop and basic signal handling.

3. Initialization function parses through all of the arguments and strips out any it recognizes.

4. It is important to call gtk_init() before any other function calls to the GTK+ libraries. Otherwise, your application will not function properly and will likely crash.

5. The gtk_init() function will terminate your application if it is unable to initialize the GUI or has any other significant problems that cannot be resolved.

Widget Hierarchy

1. *gtk_window_new()* returns a pointer to a GtkWidget. This is because every widget in GTK+ is actually a GtkWidget itself.

   GtkWidget* gtk_window_new (GtkWindowType type);

2. Widgets in GTK+ use the GObject hierarchy system, which allows you to derive new widgets from those that already exist.

3. Widget hierarchy in GTK+ is a singly inherited system, which means that each child can have only one direct parent.

- GObject is the fundamental type providing common attributes for all libraries based on it including GTK+ and Pango.

- GInitiallyUnowned should never be accessed by the programmer, since all of its members are private. It exists so that references can be floating. A floating reference is one that is not owned by anyone.

- GtkObject is the base class for all GTK+ objects. It was replaced as the absolute base class of all objects in GTK+ 2.0.

- GtkWidget is an abstract base class for all GTK+ widgets. It introduces style properties and standard functions that are needed by all widgets. The standard practice is to store all widgets as a GtkWidget.

- GtkContainer is an abstract class that is used to contain one or more widgets. It is an extremely important structure, since you could not add any other widgets to a window without it.You can cast an object as a GtkContainer with GTK_CONTAINER().

- GtkBin is another abstract class that allows a widget to contain only one child. It allows multiple widgets to have this functionality without the need for reproduction of code. You can cast an object as a GtkBin with GTK_BIN().

- GtkWindow is the standard window object you saw in program 1. You can use GTK_WINDOW() to cast an object.

NOTES ON THE simple.c GTK PROGRAM

1. We passed *GTK_WINDOW_TOPLEVEL* to gtk_window_new(). This tells GTK+ to create a new top-level window. Top-level windows use window manager decorations, have a border frame, and allow themselves to be placed by the window manager.

2. You can use *GTK_WINDOW_POPUP* to create a pop-up window, although its name is somewhat misleading in GTK+.

3. Pop-up windows are ignored by the window manager, and therefore, they have no decorations or border frame.

4. GTK_WINDOW_TOPLEVEL and GTK_WINDOW_POPUP are the only two elements available in the GtkWindowType enumeration. You will want to use GTK_WINDOW_TOPLEVEL, unless there is a compelling reason not to.

5. You should not use GTK_WINDOW_POPUP if you only want window manager decorations turned off for the window. Use *gtk_window_set_decorated (GtkWindow *window, gboolean show)* to turn off window decorations.

6. Since gtk_window_set_title() requires a GtkWindow object as it's the first parameter, we must cast our window using the GTK_WINDOW() function.

*void gtk_window_set_title (GtkWindow *window, const gchar *title);*

7. The second parameter of gtk_window_set_title() is the title that will be displayed by the window. It uses **GLib's** implementation of char, which is called gchar. When you see a parameter listed as gchar*, it will also accept const char*, because gchar* is defined as a typedef of the standard C string object.

8. The last function of interest in this section is gtk_widget_show(), which tells GTK+ to set the specified widget as visible. The widget may not be immediately shown when you call gtk_widget_show(), because GTK+ queues the widget until all preprocessing is complete before it is drawn onto the screen.

9. In addition to showing a widget, it is also possible to use gtk_widget_hide() to hide a widget from the user's view.

*void gtk_widget_hide (GtkWidget *widget);*

10. After all initialization is complete and necessary signals are connected in a GTK+ application, there will come a time when you want to let the GTK+ main loop take control and start processing events. To do this, you will call *gtk_main()*, which will continue to run until you call gtk_main_quit() or the application terminates. This should be the last GTK+ function called in main().

11. After you call gtk_main(), it is not possible to regain control of the program until a callback function is initialized. In GTK+, signals and callback functions are triggered by user actions such as button clicks, asynchronous input-output events, programmable timeouts, and others.

Using GCC and pkg-config to Compile

You run the following command from a terminal:

*gcc -Wall -g helloworld.c -o helloworld `pkg-config --cflags gtk+-2.0` *
`pkg-config --libs gtk+-2.0`

1. The -Wall option enables all types of compiler warnings. While this may not always be desirable, it can help you detect simple programming errors as you begin programming with GTK+.

2. Debugging is enabled with -g, so that you will be able to use your compiled application with GDB or your debugger of choice.

3. *pkg-config --cflags gtk+-2.0*, returns directory names to the com-piler's include path. This will make sure that the GTK+ header files are available to the compiler. Try running pkg-config --cflags gtk+-2.0 in your terminal to see an example of what is being output to the compiler.

4. *pkg-config --libs gtk+-2.0*, appends options to the command line used by the linker including library directory path extensions and a list of libraries needed for linking to the executable.

Signals and Callbacks

1. GTK+ is a system that relies on signals and callback functions.

2. A signal is a notification to your application that the user has performed some action.

3. You can tell GTK+ to run a function when the signal is emitted. These are named callback functions.

Connecting the Signal

1. The first instance of a signal you have encountered was in *sample.c*.

2. The GtkWindow was connected to the destroy() callback function. This function will be called when the destroy signal is emitted.

g_signal_connect (G_OBJECT (window), "destroy", G_CALLBACK (destroy), NULL);

3. There are four parameters to every g_signal_connect() call.

4. The first is the widget that is to be monitored for the signal.

5. Next, you specify the name of the signal you want to track. Each widget has many possible signals, all of which can be found in the API documentation.

6. The third parameter in g_signal_connect() is the callback function that will be called when the signal is emitted, cast with G_CALLBACK( ).

Callback Functions

Callback functions specified in g_signal_connect() will be called when the signal is emitted on the widget to which it was connected. For all signals, with the exception of events, which will be covered in the next section, callback functions are in the following form.

```
static void callback_function (GtkWidget *widget,
                               ... /* Other Possible Arguments */ ...,
                               gpointer data);
```

Emitting and Stopping Signals

1. By using g_signal_emit_by_name(), you can emit a signal on an object by using its textual name.

```
void g_signal_emit_by_name (gponter instance,
                            const gchar *signal_name,
                            ...);
```

2. The last parameters of g_signal_emit_by_name() are a list of parameters that should be passed to the signal and the location to store the return value. The return value can safely be ignored if it is a void function.

3. You can also use g_signal_stop_emission_by_name() to stop the current emission of a signal. This allows you to temporarily disable a signal that will be emitting because of some action performed by your code.

```
void g_signal_stop_emission_by_name (gpointer instance, const gchar *signal_name);
```

Events

1. Events are special types of signals that are emitted by the X Window System.

2. They are initially emitted by the X Window System and then sent from the window manager to your application to be interpreted by the signal system provided by GLib.

3. For example, the destroy signal is emitted on the widget, but the delete-event event is first recognized by the underlying GdkWindow of the widget and then emitted as a signal of the widget.

4.