Getting Started

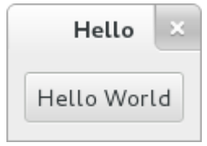
GTK is a widget toolkit. Each user interface created by GTK consists of widgets. This is implemented in C using GObject, an object-oriented framework for C. Widgets are organized in a hierarchy. The window widget is the main container. The user interface is then built by adding buttons, drop-down menus, input fields, and other widgets to the window. If you are creating complex user interfaces it is recommended to use GtkBuilder and its GTK-specific markup description language, instead of assembling the interface manually.

GTK is event-driven. The toolkit listens for events such as a click on a button, and passes the event to your application.

Below are some examples to get you started with GTK programming.

> *Note: We assume that you have GTK, its dependencies and a C compiler installed and ready to use. If you need to build GTK itself first, refer to the Compiling the GTK libraries section in this reference.*

Hello World App in C



To begin our introduction to GTK, we'll start with a simple `Hello World` GTK application.

Create a new file with the following content named `hello-world-gtk.c`.

```
#include <gtk/gtk.h>

static void
print_hello (GtkWidget *widget,
             gpointer   data)
{
  g_print ("Hello World\n");
}

static void
activate (GtkApplication *app,
          gpointer        user_data)
{
  GtkWidget *window;
  GtkWidget *button;

  window = gtk_application_window_new (app);
  gtk_window_set_title (GTK_WINDOW (window), "Window");
  gtk_window_set_default_size (GTK_WINDOW (window), 200, 200);

  button = gtk_button_new_with_label ("Hello World");
  g_signal_connect (button, "clicked", G_CALLBACK (print_hello), NULL);
  gtk_window_set_child (GTK_WINDOW (window), button);
  gtk_window_present (GTK_WINDOW (window));
}

int
main (int    argc,
      char **argv)
{
  GtkApplication *app;
  int status;

  app = gtk_application_new ("org.gtk.example", G_APPLICATION_FLAGS_NONE);
  g_signal_connect (app, "activate", G_CALLBACK (activate), NULL);
  status = g_application_run (G_APPLICATION (app), argc, argv);
  g_object_unref (app);

  return status;
}
```

You can compile the program above with GCC using:

```
gcc -o hello-world-gtk hello-world-gtk.c `pkg-config --cflags --libs gtk4`
```

For more information on how to compile a GTK application, please refer to the Compiling GTK Applications section in the GTK API reference.

Explanation

Initialising the App

In a GTK application, the purpose of the `main()` function is to create a `GtkApplication` object and run it. In this example a `GtkApplication` instance is created and initialized using `gtk_application_new()`.

When creating a `GtkApplication` you need to pick an application identifier (a name) and input to `gtk_application_new()` as parameter. For this example, `org.gtk.example` is used but for choosing an identifier for your application see this guide. Lastly `gtk_application_new()` takes a `GApplicationFlags` argument, which control some of the capabilities that your application has, like being able to open files specified on the command line, or parsing command line options.

Next the activate signal is connected to the `activate()` function above the `main()` functions. The activate signal will be sent when your application is launched with `g_application_run()` on the line below. The `gtk_application_run()` also takes as arguments the pointers to the command line arguments counter and string array; this allows GTK to parse specific command line arguments that control the behavior of GTK itself. The parsed arguments will be removed from the array, leaving the unrecognized ones for your application to parse.

Within `g_application_run` the `activate()` signal is sent and we then proceed into the `activate()` function of the application. Inside the `activate()` function we want to construct our GTK window, so that a window is shown when the application is launched. The call to `gtk_application_window_new()` will create a new `GtkApplicationWindow` instance and store it inside the `window` pointer. The window will have a frame, a title bar, and window controls depending on the platform.

A window title is set using `gtk_window_set_title()`. This function takes a `GtkWindow` pointer and a string as input. As our window pointer is a GtkWidget pointer, we need to cast it to `GtkWindow`. But instead of casting window via the usual C cast operator `(GtkWindow *)`, `window` should be cast using the `GTK_WINDOW()` macro. `GTK_WINDOW()` will perform a run time check if the pointer is an instance of the `GtkWindow` class, before casting, and emit a warning if the check fails. More information about this convention can be found here.

Finally the window size is set using `gtk_window_set_default_size()` and the window is then shown by GTK via `gtk_window_present()`.

When you exit the window, by for example pressing the X, the `g_application_run()` in the main loop returns with a number which is saved inside an integer named “status”. Afterwards, the `GtkApplication` object is freed from memory with `g_object_unref()`. Finally the status integer is returned to the operating system, and the GTK application exits.

Adding Button

As seen above, `hello-world-gtk.c` adds a button to our window, with the label “Hello World”. A new `GtkWidget` pointer is declared to accomplish this, `button`, and is initialized by calling `gtk_button_new_with_label()`, which returns a `GtkButton` to be stored inside `button`. Afterwards `button` is added to our `window`. Using `g_signal_connect` the button is connected to a function in our app called `print_hello()`, so that when the button is clicked, GTK will call this function. As the `print_hello()` function does not use any data as input, NULL is passed to it. `print_hello()` calls `g_print()` with the string “Hello World” which will print Hello World in a terminal if the GTK application was started from one.

Next steps

The GTK documentation contains a full example on how to create a complex application, capable of opening files, storing and loading settings, using menus and more complex widgets.

- FAQs
- Report a Bug
- Request a feature

COMMUNITY

- Community
- Development Blog
- Discussion
- Code of Conduct
- Donate

About the Site

- Hosted on Gitlab.
- Powered by Jekyll.
- Report an issue.

-
-
-