



EBook Gratis

APRENDIZAJE

gtk3

Free unaffiliated eBook created from
Stack Overflow contributors.

#gtk3

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con gtk3.....	2
Observaciones.....	2
Versiones.....	2
Examples.....	2
Instalación o configuración.....	3
Pitón.....	3
C ++.....	3
[C ++] "Hola mundo" en gtkmm.....	4
[C] "Hola mundo" en Gtk +.....	5
kit de inicio.....	6
Capítulo 2: GTK + 3 con Python.....	8
Examples.....	8
Una simple ventana de GTK.....	8
Enlace simple al evento de pulsación de tecla de un widget.....	8
Incrustar un video en una ventana de Gtk en Python3.....	10
Capítulo 3: GTK + 3 con Vala.....	12
Examples.....	12
Hola Mundo.....	12
El código.....	12
Compilación y Ejecución en Linux.....	12
Capítulo 4: gtk + 3 linux c.....	13
Introducción.....	13
Examples.....	13
css en acción.....	13
muestra de glarea.....	15
Capítulo 5: Gtk3 con Ruby.....	20
Examples.....	20
Levantarse y correr.....	20
Capítulo 6: Instalación de GTK + 3 en Windows (usando el repositorio GIT de GNOME) (GCC as	

Examples.....	21
Descargando GTK + 3 (también adecuado para otras versiones) y configurando.....	21
Capítulo 7: Usando Glade con Builder API.....	25
Examples.....	25
[C ++] usando Gtk :: Builder en gtkmm.....	25
Visión general.....	25
Flujo de trabajo.....	25
Ejemplo.....	25
Usando Gio :: Recurso.....	29
Creditos.....	31

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [gtk3](#)

It is an unofficial and free gtk3 ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gtk3.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con gtk3

Observaciones

GTK + 3, también conocido como Gtk3, es un conjunto de herramientas GUI multiplataforma, está escrito en C pero tiene enlaces para muchos lenguajes como C ++, Python, Vala y Ruby. (Para la lista completa vea [el sitio web de Gtk](#)).

Gtk + es parte del [Proyecto GNU](#) y está bajo las licencias GNU LGPL, lo que significa que todos los desarrolladores pueden utilizarlo, incluidos aquellos que desarrollan software privativo, sin ningún tipo de derechos de licencia o derechos de autor.

Créditos: basado libremente en <http://www.gtk.org/>

Versiones

Versión	Fecha de lanzamiento
3.20	2016-03-01
3.18	2015-09-01
3.16	2015-03-01
3.14	2014-09-01
3.12	2014-03-01
3.10	2013-09-01
3.8	2013-03-01
3.6	2012-09-01
3.4	2012-04-01
3.2	2011-10-01
3.0	2010-02-01

Fuentes:

- <https://wiki.gnome.org/Projects/GTK%2B/OldRoadmap>
-

Examples

Instalación o configuración

Pitón

Windows

La forma más fácil de instalar GTK3 para Python es usar [PyGObject para Windows](#) . Ofrece un instalador que instala la mayoría de las cosas que necesita para desarrollar aplicaciones GTK.

La cantidad de opciones que ofrece el instalador de PyGObject puede ser desalentadora, pero para la mayoría de los proyectos GTK, la única opción que debe seleccionar es `GTK+ 3.xx`

C ++

La unión de C ++ para Gtk + se conoce como **gtkmm** .

Windows

En Microsoft Windows gtkmm puede instalarse a través del entorno [MSYS2](#) . Una vez que se haya configurado el entorno MSYS2 instalando el instalador y actualizando la lista de paquetes, instale gtkmm con

```
pacman -S mingw-w64-x86_64-gtkmm3 #64 bit
pacman -S mingw-w64-i686-gtkmm3 #32 bit
```

Instale *pkg-config* para obtener fácilmente las compilaciones de compilador y vinculador y la integración de la compilación de autotools de GNU

```
pacman -S pkg-config
```

Ahora la aplicación gtkmm se puede compilar, vincular y ejecutar desde el entorno MSYS2.

```
# enable C++ 14 support if needed
# -mwindows flag is to suppress the background command-prompt window
# for GUI applications
g++ -mwindows -std=c++14 -o app.exe app.cpp `pkg-config --cflags --libs gtkmm-3.0`
./app.exe
```

Pero el ejecutable no se ejecutará fuera del shell MSYS2 debido a que faltan variables de entorno estándar para la búsqueda de `.dll`. Los siguientes `.dll` deben copiarse desde `<MSYS2 INSTALLATION DIRECTORY>\mingw64\lib\` (para la instalación de 64 bits) en el *directorio de la aplicación* (donde se encuentra el `.exe`) manualmente. Los números de versión pueden cambiar según la instalación.

```
libatk-1.0-0.dll
libatkmm-1.6-1.dll
libbz2-1.dll
libcairo-2.dll
libcairo-gobject-2.dll
libcairomm-1.0-1.dll
```

```
libepoxy-0.dll
libexpat-1.dll
libffi-6.dll
libfontconfig-1.dll
libfreetype-6.dll
libgcc_s_seh-1.dll
libgdk_pixbuf-2.0-0.dll
libgdk-3-0.dll
libgdkmm-3.0-1.dll
libgio-2.0-0.dll
libgiomm-2.4-1.dll
libglib-2.0-0.dll
libglibmm-2.4-1.dll
libgmodule-2.0-0.dll
libgobject-2.0-0.dll
libgtk-3-0.dll
libgtkmm-3.0-1.dll
libharfbuzz-0.dll
libiconv-2.dll
libintl-8.dll
libpango-1.0-0.dll
libpangocairo-1.0-0.dll
libpangoft2-1.0-0.dll
libpangomm-1.4-1.dll
libpangowin32-1.0-0.dll
libpixman-1-0.dll
libpng16-16.dll
libsigc-2.0-0.dll
libstdc++-6.dll
libwinpthread-1.dll
zlib1.dll
```

Después de este paso el programa debería ejecutarse. Pero no encontrará conjuntos de iconos estándar para Gtk +, es decir, el *tema de iconos de Adwaita*, por lo que es posible que los iconos no se carguen. Los iconos y algunos otros archivos deben copiarse en el directorio de la aplicación para que la aplicación pueda cargarlos.

Desde <MSYS2 INSTALL DIRECTORY>

```
mingw64
|
+-- lib
   |
   +-- gdk-pixbuf-2.0
share
|
+-- icons
   |
   +-- Adwaita
   |
   +-- hicolor (fallback icon theme for Gtk+)
```

A directorio de aplicaciones, con la misma estructura de directorios.

[C ++] "Hola mundo" en gtkmm

```

#include <gtkmm/application.h>
#include <gtkmm/applicationwindow.h>
#include <gtkmm/button.h>

// main window of the application
class HelloWorldWindow : public Gtk::ApplicationWindow {
    // a simple push button
    Gtk::Button btn;
public:
    HelloWorldWindow()
    : btn("Click me!") { // initialize button with a text label
        // when user presses the button "clicked" signal is emitted
        // connect an event handler for the signal with connect()
        // which accepts lambda expression, among other things
        btn.signal_clicked().connect(
            [this]() {
                btn.set_label("Hello World");
            });
        // add the push button to the window
        add(btn);
        // make the window visible
        show_all();
    }
};

int main(int argc, char *argv[]) {
    // This creates an Gtk+ application with an unique application ID
    auto app = Gtk::Application::create(argc, argv, "org.gtkmm.example.HelloApp");
    HelloWorldWindow hw;
    // this starts the application with our window
    // close the window to terminate the application
    return app->run(hw);
}

```

[C] "Hola mundo" en Gtk +

```

#include <gtk/gtk.h>

// callback function which is called when button is clicked
static void on_button_clicked(GtkButton *btn, gpointer data) {
    // change button label when it's clicked
    gtk_button_set_label(btn, "Hello World");
}

// callback function which is called when application is first started
static void on_app_activate(GApplication *app, gpointer data) {
    // create a new application window for the application
    // GtkApplication is sub-class of GApplication
    // downcast GApplication* to GtkApplication* with GTK_APPLICATION() macro
    GtkWidget *window = gtk_application_window_new(GTK_APPLICATION(app));
    // a simple push button
    GtkWidget *btn = gtk_button_new_with_label("Click Me!");
    // connect the event-handler for "clicked" signal of button
    g_signal_connect(btn, "clicked", G_CALLBACK(on_button_clicked), NULL);
    // add the button to the window
    gtk_container_add(GTK_CONTAINER(window), btn);
    // display the window
    gtk_widget_show_all(GTK_WIDGET(window));
}

```



```

int main(int argc, char *argv[]) {
    // create new GtkApplication with an unique application ID
    GtkApplication *app = gtk_application_new(
        "org.gtkmm.example.HelloApp",
        G_APPLICATION_FLAGS_NONE
    );
    // connect the event-handler for "activate" signal of GApplication
    // G_CALLBACK() macro is used to cast the callback function pointer
    // to generic void pointer
    g_signal_connect(app, "activate", G_CALLBACK(on_app_activate), NULL);
    // start the application, terminate by closing the window
    // GtkApplication* is upcast to GApplication* with G_APPLICATION() macro
    int status = g_application_run(G_APPLICATION(app), argc, argv);
    // deallocate the application object
    g_object_unref(app);
    return status;
}

```

kit de inicio

```

#include <gtk/gtk.h>
static void destroy(GtkWidget *widget, gpointer data)
{
    gtk_main_quit();
}
int main(int argc, char *argv[])
{
    gtk_init(&argc, &argv);
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Window");
    g_signal_connect(window, "destroy", G_CALLBACK(destroy), NULL);

    GtkWidget *k;
    k = gtk_fixed_new();
    gtk_container_add(GTK_CONTAINER(window), k);

    GtkWidget* la,*r;
    la = gtk_button_new_with_label ("mkl");
    gtk_fixed_put (GTK_FIXED (k), la,50,237);
    gtk_widget_set_size_request(la, 98, 90);
    //    gtk_container_set_border_width(GTK_CONTAINER (la)    , 5);

    r = gtk_button_new_with_label ("kii");
    gtk_fixed_put (GTK_FIXED (k), r,150,237);
    gtk_widget_set_size_request(r, 98, 90);

    gtk_widget_set_size_request (GTK_WIDGET(window), 300, 349);
    gtk_widget_show_all (GTK_WIDGET(window));

    gtk_main();
    return 0;
}

```

compilar:

```

c++ starterkit.c `pkg-config --libs --cflags gtk+-3.0` -o p

```

y

./p

Lea Empezando con gtk3 en línea: <https://riptutorial.com/es/gtk3/topic/4255/empezando-con-gtk3>

Capítulo 2: GTK + 3 con Python

Examples

Una simple ventana de GTK.

Simplemente presentar una ventana es fácil con GTK y Python. El siguiente ejemplo se basa en el [Tutorial de Python GTK3](#), que debe leer si es un principiante en la programación de GUI o GTK.

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk

# Set up the Gtk window
win = Gtk.Window()

# Tell Gtk what to do when the window is closed (in this case quit the main loop)
win.connect("delete-event", Gtk.main_quit)

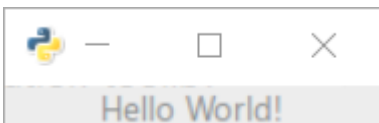
# Create a label saying Hello World!
label = Gtk.Label(label="Hello World!")

# Add the label to the window
win.add(label)

# Tell Gtk to show all widgets inside the window
win.show_all()

# Start the Gtk main loop, which returns when Gtk.main_quit is called
Gtk.main()
```

Lo que resultará (en Windows 10) en:



Enlace simple al evento de pulsación de tecla de un widget

La forma más sencilla de obtener un controlador de eventos al presionar una tecla es conectar el controlador a la señal de `key-press-event`. En este ejemplo, nos registramos para el evento para toda la ventana, pero también puede registrarse para widgets individuales.

La parte más importante es la conexión del controlador al evento:

```
self.connect("key-press-event", self.on_key_press_event)
```

En el controlador de eventos, el widget y el evento de pulsación de tecla se pasan como parámetros. Los modificadores de pulsación de tecla, como la tecla `Ctrl`, están disponibles en

`event.state` y la tecla presionada es `event.keyval` .

Los valores de las claves modificadoras se encuentran en `Gdk.ModifierType` e incluyen `CONTROL_MASK` , `SHIFT_MASK` y varios otros.

Los valores clave se encuentran en `Gdk` con los prefijos de `KEY_` , por ejemplo, la clave `h` es `Gdk.KEY_h`) Estos se pueden convertir en una cadena usando `Gdk.keyval_name()` .

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk
from gi.repository import Gdk

class MyWindow(Gtk.Window):

    key = Gdk.KEY_h

    def __init__(self):
        # init the base class (Gtk.Window)
        super().__init__()

        # state affected by shortcuts
        self.shortcut_hits = 0

        # Tell Gtk what to do when the window is closed (in this case quit the main loop)
        self.connect("delete-event", Gtk.main_quit)

        # connect the key-press event - this will call the keypress
        # handler when any key is pressed
        self.connect("key-press-event", self.on_key_press_event)

        # Window content goes in a vertical box
        box = Gtk.VBox()

        # mapping between Gdk.KEY_h and a string
        keyname = Gdk.keyval_name(self.key)

        # a helpful label
        instruct = Gtk.Label(label="Press Ctrl+%s" % keyname)
        box.add(instruct)

        # the label that will respond to the event
        self.label = Gtk.Label(label="")
        self.update_label_text()

        # Add the label to the window
        box.add(self.label)

        self.add(box)

    def on_key_press_event(self, widget, event):

        print("Key press on widget: ", widget)
        print("          Modifiers: ", event.state)
        print("          Key val, name: ", event.keyval, Gdk.keyval_name(event.keyval))

        # check the event modifiers (can also use SHIFTMASK, etc)
        ctrl = (event.state & Gdk.ModifierType.CONTROL_MASK)
```

```

# see if we recognise a keypress
if ctrl and event.keyval == Gdk.KEY_h:
    self.shortcut_hits += 1
    self.update_label_text()

def update_label_text(self):
    # Update the label based on the state of the hit variable
    self.label.set_text("Shortcut pressed %d times" % self.shortcut_hits)

if __name__ == "__main__":
    win = MyWindow()
    win.show_all()

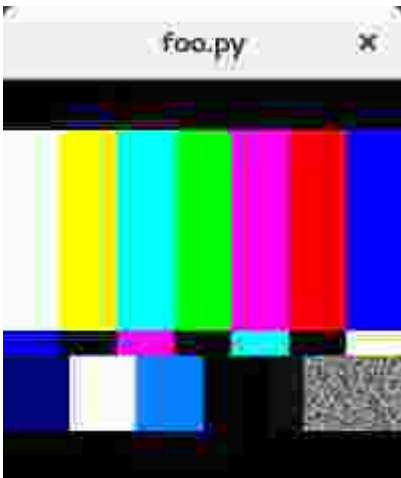
# Start the Gtk main loop
Gtk.main()

```

Se puede lograr un comportamiento más avanzado para los accesos directos de toda la aplicación con un grupo acelerador (`Gtk.AccelGroup`), pero a menudo todo lo que necesita para capturar los eventos de teclado que desea para un widget específico es un controlador de pulsación rápida de teclas.

Incrustar un video en una ventana de Gtk en Python3

A continuación se muestra un ejemplo de canalización de Gstreamer incrustada en una ventana gtk simple. Cuando se ejecuta, una pequeña ventana debe aparecer así:



```

import gi
gi.require_version('Gtk', '3.0')
gi.require_version('Gst', '1.0')

from gi.repository import Gtk, Gst
Gst.init(None)
Gst.init_check(None)

class GstWidget(Gtk.Box):
    def __init__(self, pipeline):
        super().__init__()
        # Only setup the widget after the window is shown.
        self.connect('realize', self._on_realize)

        # Parse a gstreamer pipeline and create it.

```

```

self._bin = Gst.parse_bin_from_description(pipeline, True)

def _on_realize(self, widget):
    pipeline = Gst.Pipeline()
    factory = pipeline.get_factory()
    gtxsink = factory.make('gtxsink')
    pipeline.add(self._bin)
    pipeline.add(gtxsink)
    # Link the pipeline to the sink that will display the video.
    self._bin.link(gtxsink)
    self.pack_start(gtxsink.props.widget, True, True, 0)
    gtxsink.props.widget.show()
    # Start the video
    pipeline.set_state(Gst.State.PLAYING)

window = Gtk.ApplicationWindow()

# Create a gstreamer pipeline with no sink.
# A sink will be created inside the GstWidget.
widget = GstWidget('videotestsrc')
widget.set_size_request(200, 200)

window.add(widget)

window.show_all()

def on_destroy(win):
    Gtk.main_quit()

window.connect('destroy', on_destroy)

Gtk.main()

```

Lea GTK + 3 con Python en línea: <https://riptutorial.com/es/gtk3/topic/4257/gtk-plus-3-con-python>

Capítulo 3: GTK + 3 con Vala

Examples

Hola Mundo

Podría ser aún más básico, pero esto muestra algunas de las características del lenguaje Vala.

El código

```
using Gtk;

int main (string[] args) {
    Gtk.init (ref args);

    var window = new Window ();
    window.title = "First GTK+ Program";
    window.border_width = 10;
    window.window_position = WindowPosition.CENTER;
    window.set_default_size (350, 70);
    window.destroy.connect (Gtk.main_quit);

    var button = new Button.with_label ("Click me!");
    button.clicked.connect (() => {
        button.label = "Thank you";
    });

    window.add (button);
    window.show_all ();

    Gtk.main ();
    return 0;
}
```

Todas las clases de GTK + están dentro del espacio de nombres de `Gtk` . Debe inicializar todos los programas GTK + con `Gtk.init ()` .

Compilación y Ejecución en Linux

```
$ valac --pkg gtk+-3.0 gtk-hello.vala
$ ./gtk-hello
```

Esto necesita el compilador `valac` , `gcc` , los paquetes de desarrollo `glib` y `gtk3` instalados en su sistema.

Tomado de la [wiki de GNOME](#) .

Lea **GTK + 3 con Vala en línea**: <https://riptutorial.com/es/gtk3/topic/4673/gtk-plus-3-con-vala>

Capítulo 4: gtk + 3 linux c

Introducción

ejemplos de código y algunas otras cosas

Examples

css en acción

```
#include <gtk/gtk.h> //jjk.c
static void destroy(GtkWidget *widget, gpointer data)
{
    gtk_main_quit();
}
int main(int argc, char *argv[])
{
    gtk_init(&argc, &argv);
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Window");
    g_signal_connect(window, "destroy", G_CALLBACK(destroy), NULL);

    GtkWidget *k;
    k = gtk_fixed_new();
    gtk_container_add(GTK_CONTAINER(window), k);

    GtkWidget* la,*r;
    la = gtk_button_new_with_label ("mkl");
    gtk_fixed_put (GTK_FIXED (k), la,50,237);
    gtk_widget_set_size_request(la, 98, 90);
    //    gtk_container_set_border_width(GTK_CONTAINER (la)    , 5);

    union {
    char w[4]={0xf,0xe,0xd,0xa};;
    uint t;
    } tc;

    GtkCssProvider *provider = gtk_css_provider_new ();
    gtk_css_provider_load_from_path (provider,
        "/home/alex/gtk-widgets.css", NULL);

    r = gtk_button_new_with_label ("kii");
    gtk_fixed_put (GTK_FIXED (k), r,150,237);
    gtk_widget_set_size_request(r, 98, 90);

    gtk_widget_set_size_request (GTK_WIDGET(window), 300, 349);

    GtkStyleContext *context;
    context = gtk_widget_get_style_context (la);
    gtk_style_context_add_provider (context,
        GTK_STYLE_PROVIDER(provider),
        GTK_STYLE_PROVIDER_PRIORITY_USER);
```



```

//      gtk_style_context_save (context);
//      gtk_style_context_add_provider_for_screen(gdk_screen_get_default(),
//
GTK_STYLE_PROVIDER(provider),TK_STYLE_PROVIDER_PRIORITY_USER);

gtk_widget_show_all(GTK_WIDGET(window));
#define h 7

printf("%xh\n",tc.t);
gtk_main();
return 0;
}

```

gtk-widgets.css

```

GtkButton:hover {
    color: yellowgreen;
    background-color: green;
    opacity: 0.95;
    text-decoration: underline;
    background-image: -gtk-gradient (linear,
        left top,
        left bottom,
        color-stop(0.0, rgba(34, 97, 70, 1)),
        color-stop(0.30, rgba(56, 145, 118, 0.9)),
        color-stop(0.81, rgba(34, 131, 116, 0.9)),
        color-stop(1.00, rgba(104, 191, 134, 1)));
}

GtkButton:active{
    color: yellowgreen;
    background-color: green;
    opacity: 0.97;
    text-decoration: underline;
    background-image: -gtk-gradient (linear,
        left top,
        left bottom,
        color-stop(0.000, rgba(104, 191, 134, 1)),
        color-stop(0.11, rgba(34, 131, 116, 1)),
        color-stop(0.32, rgba(34, 97, 70, 1)),
        color-stop(0.70, rgba(56, 145, 118, 1)),
        color-stop(0.91, rgba(34, 131, 116, 1)),
        color-stop(1.00, rgba(104, 191, 134, 1)));
}

GtkButton {
    color: yellowgreen;
    background-color: green;
    opacity: 0.797;
    text-decoration: underline;
    background-image: -gtk-gradient (linear,

```

```

        left top,
        left bottom,
        color-stop(0.0, rgba(34, 97, 70, 1)),
        color-stop(0.50, rgba(56, 145, 118, 1)),
        color-stop(0.51, rgba(34, 131, 116, 1)),
        color-stop(1.00, rgba(104, 191, 134, 1));
    }

c++ jjk.c --target=arm-linux-gnu `pkg-config --libs --cflags gtk+-3.0` -o op

```

muestra de glarea

Shaders 3.3 + extension.seen at radeon hd5500

```
#version 330
```

triángulos desplegados de modo que lo normal apunte fuera de un cubo. siga cada orden de triángulo de 3 puntos.

```

//#include <stdio.h>
//#include <string.h>
//#include <iostream>

//#include <glib.h>

#include <gdk/gdkx.h>
#include <epoxy/glx.h>
#include <epoxy/gl.h>
#include <gtk/gtk.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

const GLchar *vert_src = "\n" \
"#version 330\n" \
"    #extension GL_ARB_explicit_uniform_location : enable\n" \
"    \n" \
"    out vec4 knn;\n" \
"    layout(location = 0) in vec4 in_position;\n" \
"    layout(location =10)uniform mat4 projection;\n" \
"    out vec4 P;\n" \
"void main()\n" \
"{\n" \
"    gl_Position = projection*in_position;\n" \
"    knn=in_position ;\n" \

```

```

"                                     \n" \
"//  gl_Position = in_position; \n" \
"}                                     \n" ;
const GLchar *frag_src = "\n" \
"#version 330                                     \n" \
"                                     \n" \
"         vec4 b;                                     \n" \
"     in  vec4 knn;                                     \n" \
"void main (void)                                     \n" \
"{                                     \n" \
" if(gl_FrontFacing){                                     \n" \

"     b= vec4(gl_FragCoord.x/10.0,gl_FragCoord.y/357.0,gl_FragCoord.z*   0.5,1.0);
\n" \
"         gl_FragColor = vec4(0.021,-.1777*(knn.y+2)+0.1+1.21557*knn.x+knn.z
,1.215*knn.z+0.583*knn.y +.0357*knn.x+3*b.y, 1.0); \n" \
"     }                                     \n" \
"else discard; \n" \
"                                     \n" \
"                                     \n" \
"}                                     \n";

GLfloat t[16]={ 1,0,0,0,
               0,1,0,0,
               0,0,1,0,
               0,0,0,1};
glm::mat4 yt,b;

GLuint gl_vao, gl_buffer, gl_program;

static gboolean realise(GtkGLArea *area, GdkGLContext *context)
{

gtk_gl_area_make_current(GTK_GL_AREA(area));
if (gtk_gl_area_get_error (GTK_GL_AREA(area)) != NULL)
{
printf("failed to initialiize buffers\n");
return false;
}

GLfloat verts[] =
{
+0.7,+0.7,+0.7,
-0.7,0.7, 0.7,
-0.7,+0.7, -0.7,

0.7,+0.7, -0.7,
+0.7,+0.7,+0.7,
-0.7,+0.7, -0.7,

-0.7,-0.7, +0.7,
+0.7,+0.7,+0.7,
-0.7,0.7, 0.7,

+0.7,+0.7,+0.7,
-0.7,-0.7, +0.7,
0.7,-0.7, 0.7,

```

```

    +0.7,-0.7,-0.7,
    -0.7,-0.7,-0.7,
    0.7,+0.7, -0.7,

    -0.7,+0.7, -0.7,
    0.7,+0.7, -0.7,
    -0.7,-0.7, -0.7,

    -0.7,-0.7, +0.7,
    -0.7,-0.7, -0.7,
    0.7,-0.7, 0.7,

+0.7,-0.7, -0.7,

0.7,-0.7, 0.7,

-0.7,-0.7, -0.7,

    0.7,+0.7, -0.7,
    +0.7,+0.7,+0.7,
+0.7,-0.7, -0.7,

    0.7,-0.7, -0.7,
    +0.7,+0.7,+0.7,

0.7,-0.7, 0.7,

    -0.7,-0.7, -0.7,
    -0.7,0.7, 0.7,
-0.7,+0.7, -0.7,

    -0.7,-0.7, +0.7,
    -0.7,0.7, 0.7,
    -0.7,-0.7, -0.7

};
b=glm::lookAt(glm::vec3(1.75,-2.38,1.4444), glm::vec3( 0., 0., 0.),glm::vec3( 0.,0.2,-00.));
yt=glm::perspective(45., 1., 1.2, 300.);
b=yt*b*glm::mat4(1.);

//b=glm::lookAt(glm::vec3(0., 0.,-1.),glm::vec3( 0., 0., 0.),glm::vec3( 0.,025.,-1.));
//yt=yt*b;

GLuint frag_shader, vert_shader;
frag_shader = glCreateShader(GL_FRAGMENT_SHADER);
vert_shader = glCreateShader(GL_VERTEX_SHADER);

```

```

glShaderSource(frag_shader, 1, &frag_src, NULL);
glShaderSource(vert_shader, 1, &vert_src, NULL);

glCompileShader(frag_shader);
glCompileShader(vert_shader);

gl_program = glCreateProgram();
glAttachShader(gl_program, frag_shader);
glAttachShader(gl_program, vert_shader);
glLinkProgram(gl_program);

// glUniformMatrix4fv(1, 1, 0, glm::value_ptr(b));

glGenVertexArrays(1, &gl_vao);
glBindVertexArray(gl_vao);

glGenBuffers(1, &gl_buffer);
glBindBuffer(GL_ARRAY_BUFFER, gl_buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(verts), verts, GL_STATIC_DRAW);

glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
glBindVertexArray(0);

return TRUE;
}

static gboolean render(GtkGLArea *area, GdkGLContext *context)
{
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_BLEND);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glClearColor(0.0, 0.0, 0.0, 1.0);

    glUseProgram(gl_program);
    glUniformMatrix4fv(10, 1, 0, &b[0][0]);

    glBindVertexArray(gl_vao);

    glDrawArrays(GL_TRIANGLES, 0, 36 );

    glBindVertexArray(0);
    glUseProgram(0);

    glFlush();

    return TRUE;
}

```

```
}

int main(int argc, char** argv)
{
    gtk_init(&argc, &argv);

    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL),
              *gl_area = gtk_gl_area_new();

    g_signal_connect(window, "delete-event", G_CALLBACK(gtk_main_quit), NULL);
    g_signal_connect(gl_area, "realize",      G_CALLBACK(realise),      NULL);
    g_signal_connect(gl_area, "render",      G_CALLBACK(render),      NULL);

    gtk_container_add(GTK_CONTAINER(window), gl_area);

    gtk_widget_show_all(window);

    gtk_main();

    return 0;
}
```

Lea gtk + 3 linux c en línea: <https://riptutorial.com/es/gtk3/topic/8246/gtk-plus-3-linux-c>

Capítulo 5: Gtk3 con Ruby

Examples

Levantarse y correr

```
# Like any other ruby code, require gtk3 after installing from "gem install gtk3"
require 'gtk3'

# Like in Rails, you import working functions from a higher class, in this case the GTK Window
class RubyApp < Gtk::Window

# Calling the original method from GTK Window and redefining the defaults
  def initialize
    super

    # Printing window title
    set_title "Center"

    # Invoking built-in GTK connection and calling destroy to replicate the quit action
    signal_connect "destroy" do
      Gtk.main_quit
    end

    # Sets the window size 500px wide by 400px tall
    set_default_size 500, 400

    # Where the window should be displayed on the screen
    set_window_position Gtk::Window::Position::CENTER

    # After initialization, show everything
    show
  end
end

# Call the class, just like any other ruby program
window = RubyApp.new

# GTK method. Runs until destroy is called
Gtk.main
```

Guarde este archivo como `new.rb` y ejecútelo desde el terminal `ruby new.rb`

Lea Gtk3 con Ruby en línea: <https://riptutorial.com/es/gtk3/topic/5501/gtk3-con-ruby>

Capítulo 6: Instalación de GTK + 3 en Windows (usando el repositorio GIT de GNOME) (GCC asumiendo el lenguaje C está instalado)

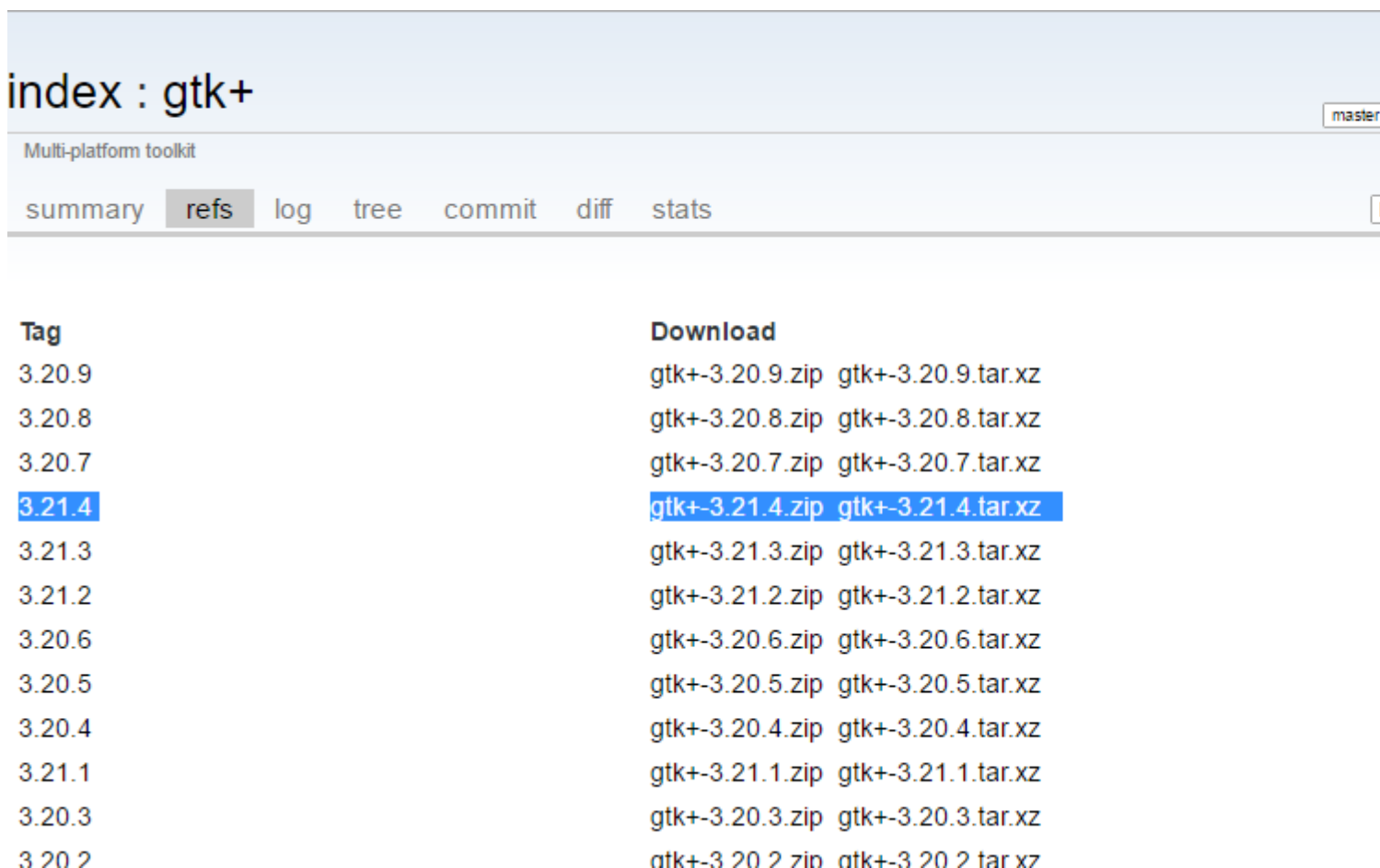
Examples

Descargando GTK + 3 (también adecuado para otras versiones) y configurando

Descargando el paquete:

La forma más fácil de descargar el paquete GTK requerido es buscarlo y descargarlo a través de este enlace: <https://git.gnome.org/browse/gtk+> (repositorio de GNOME GIT)

El repositorio GIT de GNOME proporciona los paquetes para diferentes versiones y puede encontrar fácilmente la versión deseada desplazándose por la lista. Estos son subidos por otros usuarios o autores y, a menudo, encontrarás nuevos paquetes que están por venir.



The screenshot shows the 'index : gtk+' page of the GNOME GIT repository. It includes a navigation bar with 'summary', 'refs', 'log', 'tree', 'commit', 'diff', and 'stats'. Below the navigation bar is a table with two columns: 'Tag' and 'Download'. The 'Tag' column lists versions from 3.20.9 down to 3.20.2, with 3.21.4 highlighted in blue. The 'Download' column lists corresponding zip and tar.xz files for each version, with the 3.21.4 files also highlighted in blue.

Tag	Download
3.20.9	gtk+-3.20.9.zip gtk+-3.20.9.tar.xz
3.20.8	gtk+-3.20.8.zip gtk+-3.20.8.tar.xz
3.20.7	gtk+-3.20.7.zip gtk+-3.20.7.tar.xz
3.21.4	gtk+-3.21.4.zip gtk+-3.21.4.tar.xz
3.21.3	gtk+-3.21.3.zip gtk+-3.21.3.tar.xz
3.21.2	gtk+-3.21.2.zip gtk+-3.21.2.tar.xz
3.20.6	gtk+-3.20.6.zip gtk+-3.20.6.tar.xz
3.20.5	gtk+-3.20.5.zip gtk+-3.20.5.tar.xz
3.20.4	gtk+-3.20.4.zip gtk+-3.20.4.tar.xz
3.21.1	gtk+-3.21.1.zip gtk+-3.21.1.tar.xz
3.20.3	gtk+-3.20.3.zip gtk+-3.20.3.tar.xz
3.20.2	gtk+-3.20.2.zip gtk+-3.20.2.tar.xz

Alternativas:

La otra opción o el método alternativo para adquirir el paquete es utilizar MSYS2. Este método es un poco complicado y no está bien explicado o documentado. La mejor manera es descargar directamente el paquete desde un sitio web externo de alojamiento de archivos.

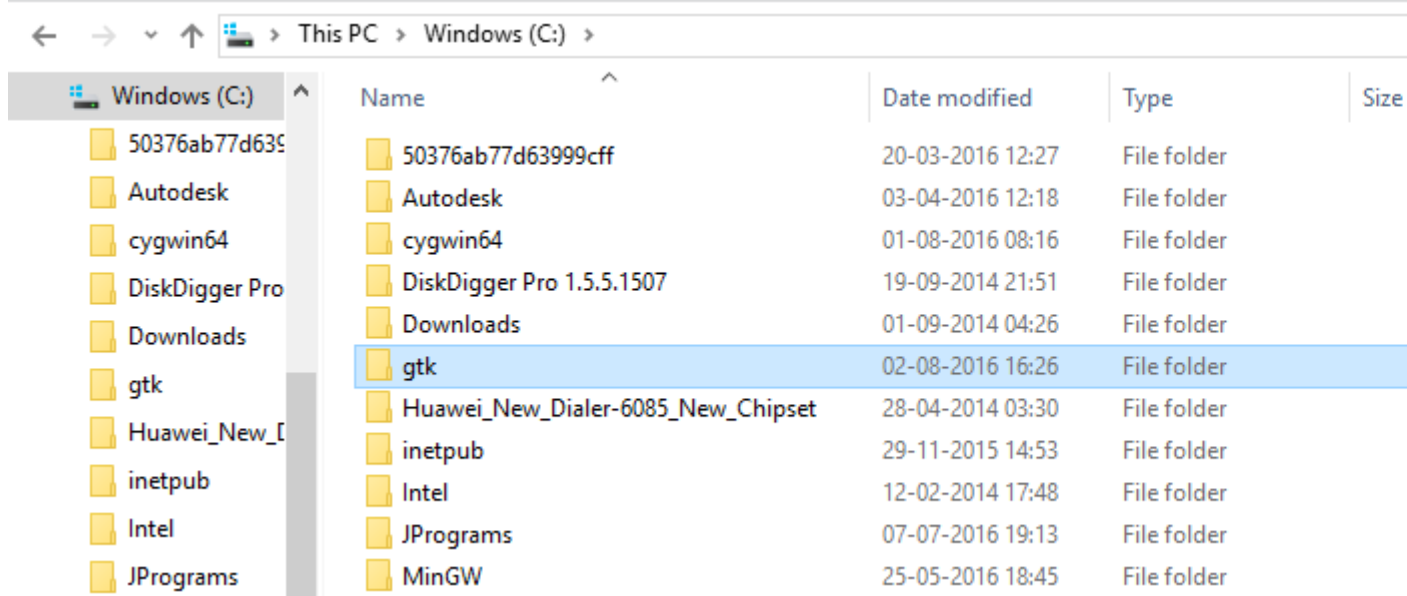
Método MSYS2-

1. <http://www.gtk.org/download/windows.php> (recurso oficial del sitio web sobre la adquisición del paquete)
2. <https://msys2.github.io/> (descargar MSYS2)

Nota: El resto de esta documentación se basa en la suposición de que el paquete GTK se descarga desde el repositorio GIT de GNOME. No probado con el método MSYS2. Dado que el paquete sigue siendo universal, el método utilizado no debería ser un problema. Basado en la facilidad, la manera altamente recomendada de obtener el paquete es T0 Usar el repositorio GIT de GNOME.

Extracción y almacenamiento del paquete en una ubicación deseada (explicado mediante la unidad C:).

1. Cree una carpeta llamada `gtk` en la `C: Drive`. Puede elegir otros nombres para la carpeta, pero este nombre nos ayuda a identificar el contenido con bastante facilidad y recordar el nombre se vuelve más fácil. Llamemos a la ubicación de la carpeta `Gtk` como `%GD%` para mayor facilidad y en este caso es `C:\gtk` (podría ser diferente en su caso).



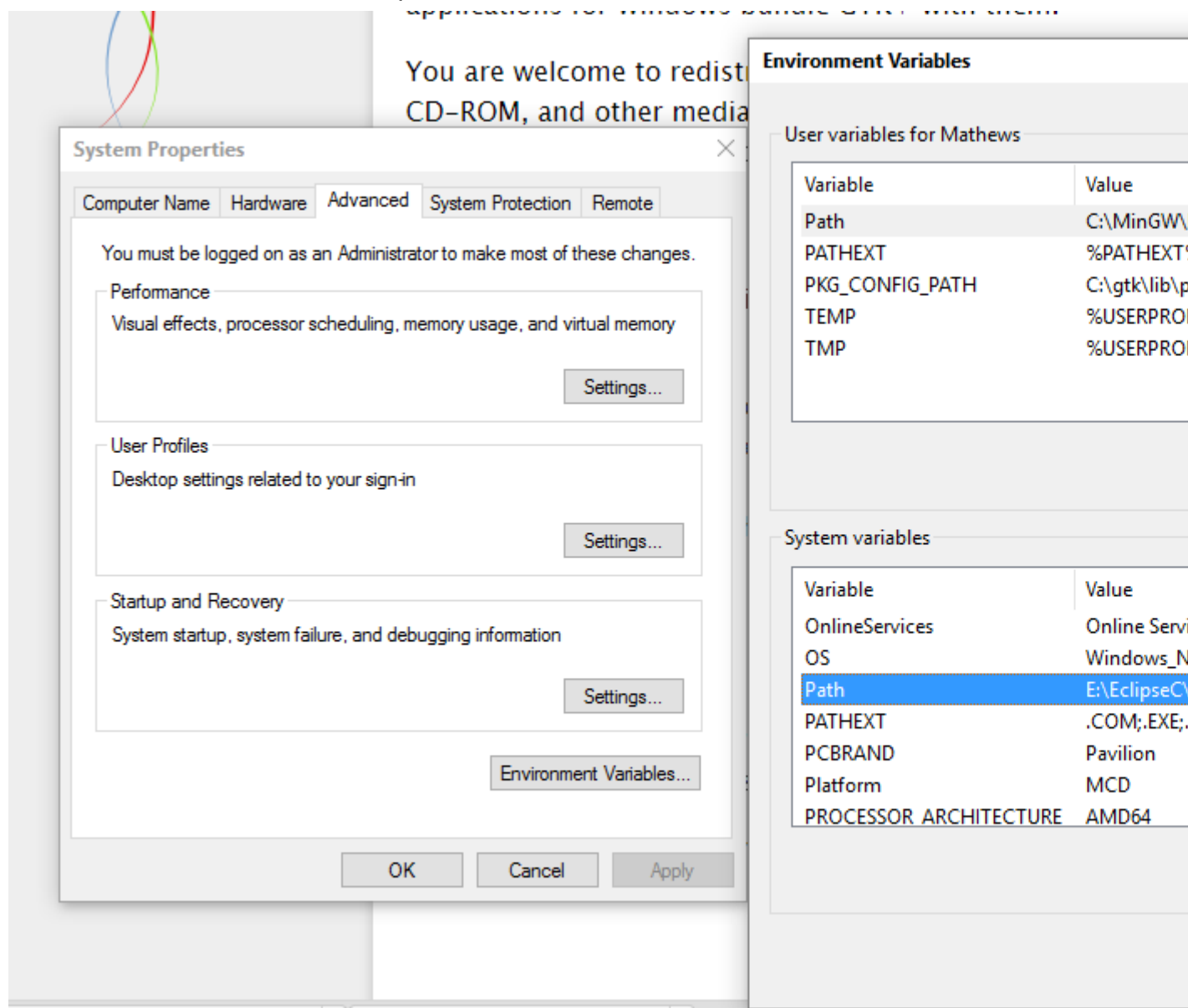
2. Configuración de la ruta del binario `Gtk` en las variables de entorno de Windows. Esto es importante para acceder y obtener las bibliotecas `Gtk` requeridas mientras se usa `Cmd` para compilar y ejecutar los programas. Para configurar la variable de entorno de ruta:

- Windows XP : right-click on "My Computer" -> "Properties" .
- Windows Vista/7/8/10 : right-click on "Computer" -> "Properties" -> "Advanced system settings". Click on "Advanced tab" -> "Environment variables".

- Agregue la nueva `Gtk bin file Path (%GD%\bin)` a la variable `Path` (debajo de las variables del sistema) o agregue una variable llamada `Path` a la sección de variables del usuario y agregue el valor en ella disponible en el cuadro de diálogo que se abre. También debe agregar una nueva variable llamada `PKG_CONFIG_PATH` y asignarle el siguiente valor `C:\gtk\lib\pkgconfig` (ya que mi carpeta `gtk` se encuentra en la unidad `C` `C:\gtk\lib\pkgconfig`).

Nombre de la variable	Valor
Camino	% GD% \ bin
PKG_CONFIG_PATH	% GD% \ lib \ pkgconfig

% GD% es la ubicación de la carpeta `Gtk`



Ahora hemos terminado con la extracción y configuración de las variables de entorno. Encontrará que he agregado el valor a la variable Ruta en las secciones de Variables del sistema, así como en la sección Variables del usuario. Agregar a cualquiera de estas secciones es más que suficiente.

3. Probar la configuración (debe tener GCC instalado en su sistema, preferiblemente MinGW (<http://www.mingw.org/>)) : ejecute los siguientes comandos en el cmd:

```
pango-querymodules: %GTKDIR%\etc\pango\pango.modules
```

```
gdk-pixbuf-query-loaders: %GTKDIR%\lib\gdk-pixbuf-2.0\2.10.0\loaders.cache
```

```
gtk-query-immodules-3.0: %GTKDIR%\lib\gtk-3.0\3.0.0\immodules.cache
```

- En la consola, verifique que "pkg-config" imprima algo razonable escribiendo: `pkg-config --cflags --libs gtk+-3.0` Debe imprimir algo similar a lo que se muestra en la siguiente imagen.

```
C:\Users\Mathews>pango-querymodules > C:\gtk\etc\pango\pango.modules
C:\Users\Mathews>pkg-config --cflags --libs gtk+-3.0
-mms-bitfields -IC:/gtk/include/gtk-3.0 -IC:/gtk/include/atk-1.0 -IC:/gtk/include/cairo -IC:/gtk/include/gdk-pixbuf-2.0
-gtk/include/pixman-1 -IC:/gtk/include/freetype2 -IC:/gtk/include/libpng15 -Wl,-luuid -LC:/gtk/lib -lgtk-3 -lgdk-3 -l
0 -lpangowin32-1.0 -lgdi32 -lfreetype -lfontconfig -lpango-1.0 -lm -lcairo -lgobject-2.0 -lglib-2.0 -lintl
C:\Users\Mathews>
```

Eso es. Ha descargado y configurado UP GTK en su sistema Windows.

Lea [Instalación de GTK + 3 en Windows \(usando el repositorio GIT de GNOME\) \(GCC asumiendo el lenguaje C está instalado\)](https://riptutorial.com/es/gtk3/topic/5602/instalacion-de-gtk-plus-3-en-windows--usando-el-repositorio-git-de-gnome---gcc-asumiendo-el-lenguaje-c-esta-instalado-) en línea: <https://riptutorial.com/es/gtk3/topic/5602/instalacion-de-gtk-plus-3-en-windows--usando-el-repositorio-git-de-gnome---gcc-asumiendo-el-lenguaje-c-esta-instalado->

Capítulo 7: Usando Glade con Builder API

Examples

[C ++] usando Gtk :: Builder en gtkmm

Visión general

Gtk + admite un flujo de trabajo en el que la tarea de diseño de la interfaz de usuario y la tarea de programación se desacoplan. Si bien los elementos de la interfaz de usuario, como botones, menús, diseño, etc., pueden agregarse directamente desde el código, este enfoque no solo desordena el código, sino que también dificulta el cambio de la interfaz de usuario para cualquier persona, excepto para el programador. Además, algunos elementos de la interfaz se usan solo para mantener la estructura del diseño y no necesitan participar en la lógica, agregarlos desde el código solo hace que sea más largo. En su lugar, se puede usar Glade para generar la descripción de la interfaz de usuario como XML y Gtk + Builder API se puede usar para cargar la interfaz de usuario y operar en ella.

Flujo de trabajo

1. Diseñar elementos de la interfaz de usuario en [Glade](#) usando arrastrar y soltar. Glade genera un archivo XML que contiene descripciones de UI. Esto también puede hacerse manualmente escribiendo la sintaxis XML adecuada y guardándola con una extensión

.glade .

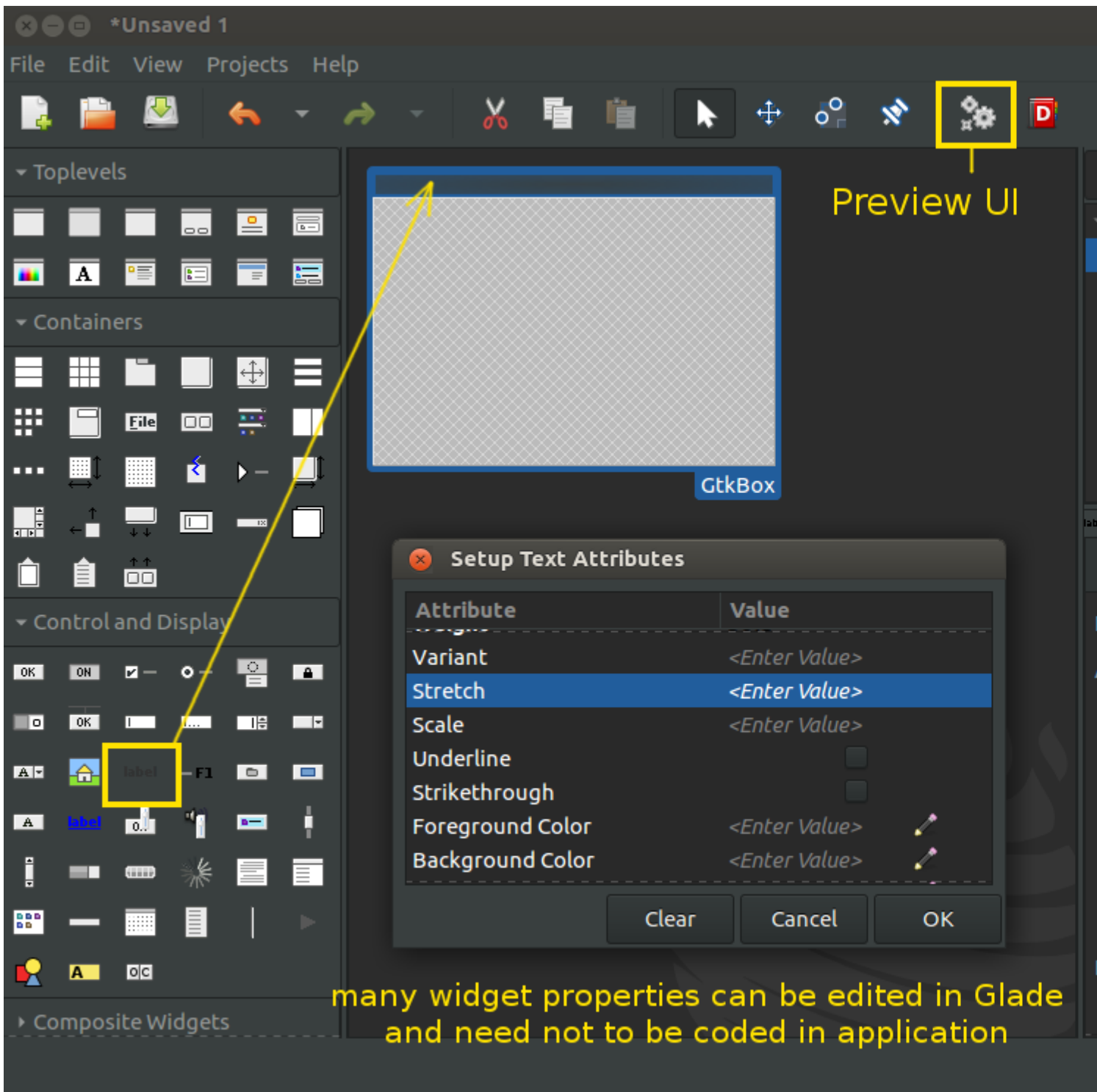
2. Cargar UI desde el archivo directamente con

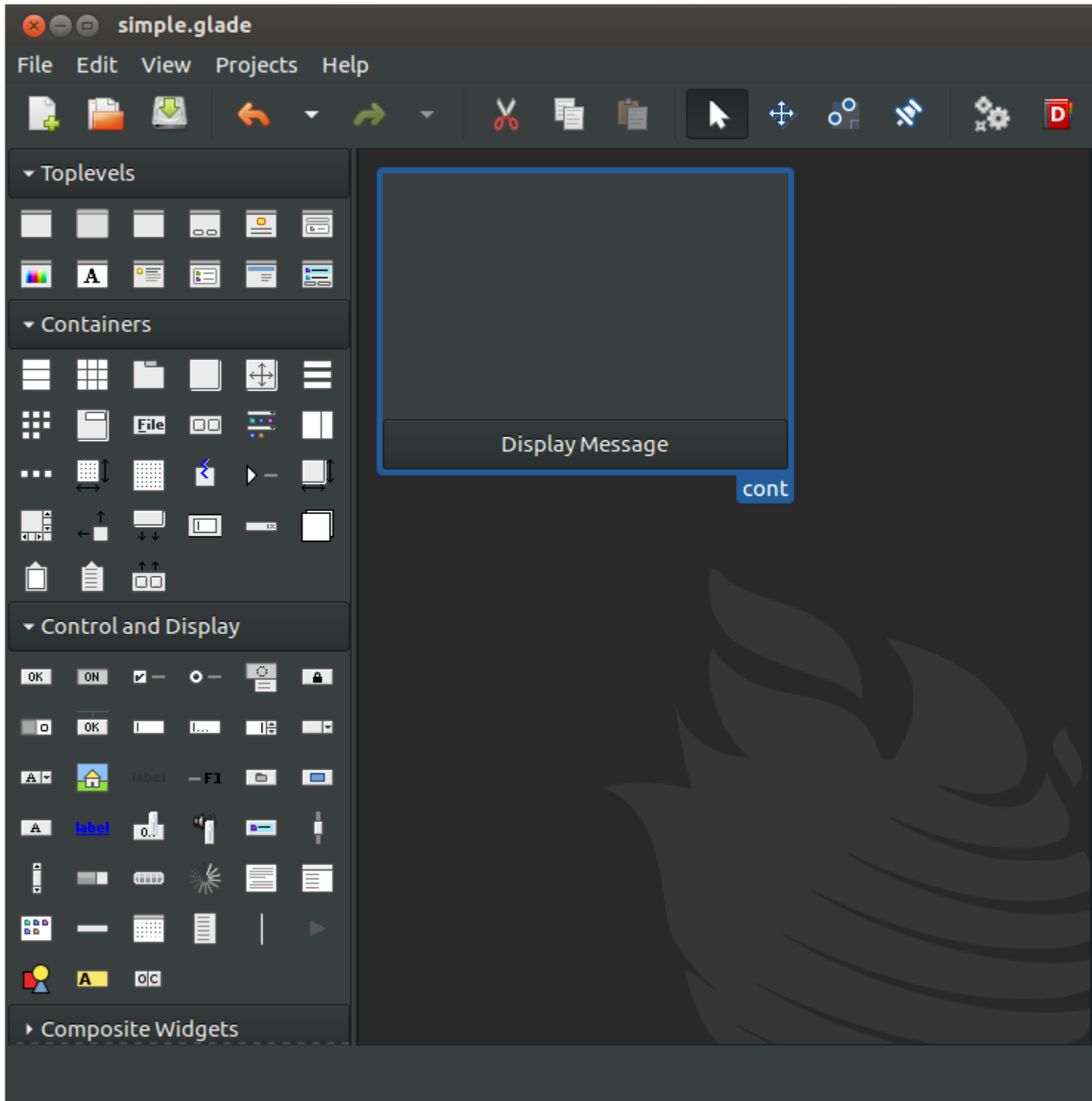
```
auto ui = Gtk::Builder::create_from_file("ui.glade");
```

3. Acceder a elementos individuales de la interfaz de usuario

```
// when element doesn't need to be added to another UI element
auto ui_elem = Glib::RefPtr<Gtk::Button>::cast_dynamic(
    ui->get_object("button_UI_id")
);
// when element needs to be added to another widget
Gtk::Button *btn = nullptr;
ui->get_widget<Gtk::Button>("button_UI_id", btn);
```

Ejemplo





simple.glade

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated with glade 3.20.0 -->
<interface>
  <requires lib="gtk+" version="3.20"/>
  <object class="GtkBox" id="cont">
    <property name="width_request">200</property>
    <property name="height_request">200</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="orientation">vertical</property>
```

```

<child>
  <object class="GtkLabel" id="display_label">
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="wrap">True</property>
    <attributes>
      <attribute name="weight" value="bold"/>
      <attribute name="scale" value="5"/>
      <attribute name="foreground" value="#a4a400000000"/>
    </attributes>
  </object>
<packing>
  <property name="expand">True</property>
  <property name="fill">True</property>
  <property name="position">0</property>
</packing>
</child>
<child>
  <object class="GtkButton" id="display_button">
    <property name="label" translatable="yes">Display Message</property>
    <property name="visible">True</property>
    <property name="can_focus">True</property>
    <property name="receives_default">True</property>
  </object>
<packing>
  <property name="expand">False</property>
  <property name="fill">True</property>
  <property name="position">1</property>
</packing>
</child>
</object>
</interface>

```

simple.cpp

```

#include <gtkmm/application.h>
#include <gtkmm/applicationwindow.h>
#include <gtkmm/button.h>
#include <gtkmm/label.h>
#include <gtkmm/box.h>
#include <gtkmm/builder.h>

class HelloWorld : public Gtk::ApplicationWindow {
  Gtk::Box *cont;
  Glib::RefPtr<Gtk::Label> display_label;
  Glib::RefPtr<Gtk::Button> display_btn;
  Glib::RefPtr<Gtk::Builder> ui;
public:
  HelloWorld()
  : ui{Gtk::Builder::create_from_file("simple.glade")} {
    if(ui) {
      ui->get_widget<Gtk::Box>("cont", cont);
      display_label = Glib::RefPtr<Gtk::Label>::cast_dynamic(
        ui->get_object("display_label")
      );
      display_btn = Glib::RefPtr<Gtk::Button>::cast_dynamic(
        ui->get_object("display_button")
      );
      if(cont && display_label && display_btn) {
        display_btn->signal_clicked().connect(

```

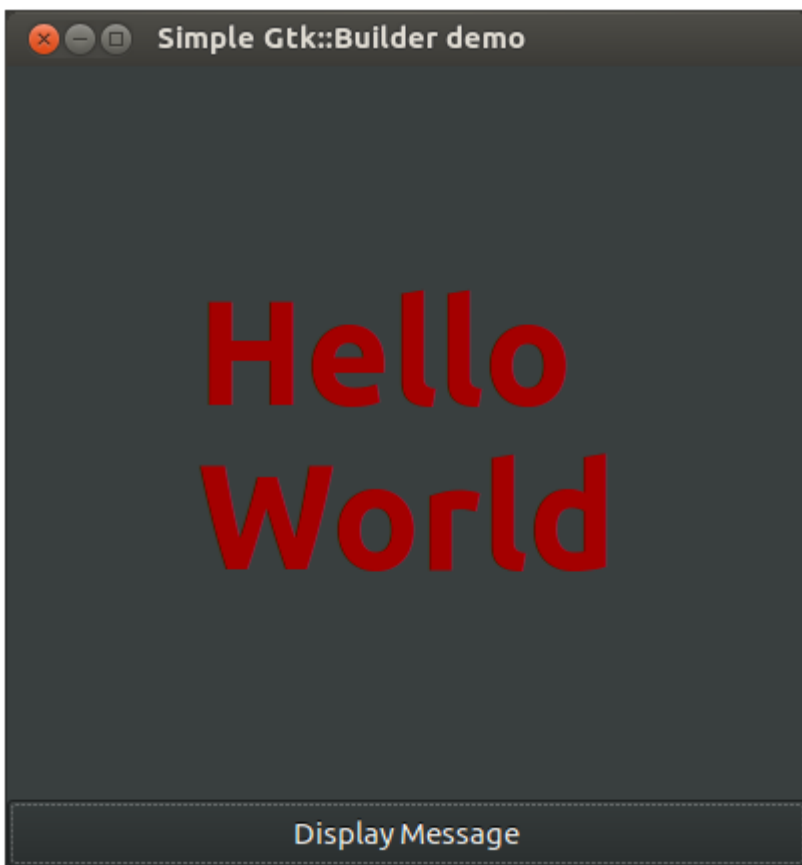
```

        [this]() {
            display_label->set_text("Hello World");
        });
        add(*cont);
    }
}
set_title("Simple Gtk::Builder demo");
set_default_size(400, 400);
show_all();
}
};

int main(int argc, char *argv[]) {
    auto app = Gtk::Application::create(
        argc, argv,
        "org.gtkmm.example.HelloApp"
    );
    HelloWorld hw;
    return app->run(hw);
}

```

Salida



Usando Gio :: Recurso

Cargar interfaces de usuario directamente desde archivos `.glade` es rápido y fácil. Pero cuando la aplicación está empaquetada, las descripciones de la interfaz de usuario, los iconos y otras imágenes se pueden agrupar en *paquetes de recursos*. Primero se debe crear una descripción

de recurso como archivo XML.

recursos.xml

```
<gresources>
  <gresource prefix="/unique/prefix/">
    <file>icon.png</file>
    <!-- text files such as XML can be compressed to save memory -->
    <file compressed="true">ui.glade</file>
  </gresource>
</gresources>
```

Luego cree un archivo `.gresource` separado o un archivo `.c` que contenga recursos como datos de cadena para vincularlos como parte de la aplicación.

```
# generates separate resource file
glib-compile-resources --target=ui.gresource resources.xml
# generates .c file
glib-compile-resources --generate-source resources.xml
```

Luego desde el código de la aplicación cargar el paquete de recursos

```
// from separate file
auto resource_bundle = Gio::Resource::create_from_file("ui.gresource");
// from stream of bytes in .c file
auto resource_bundle = Glib::wrap(draw_resource_get_resource());
resource_bundle.register_global();
```

Desde el paquete de recursos, cargar elementos de la interfaz de usuario

```
auto ui = Gtk::Builder::create_from_resource("/unique/prefix/ui.glade");
```

Lea Usando Glade con Builder API en línea: <https://riptutorial.com/es/gtk3/topic/5579/usando-glade-con-builder-api>

Creditos

S. No	Capítulos	Contributors
1	Empezando con gtk3	B8vrede , Community , Samik , Алексей Неудачин
2	GTK + 3 con Python	12345ieeee , Aaron Schif , B8vrede , Inductiveload , joaquinlpereyra , RamenChef
3	GTK + 3 con Vala	meskobalazs
4	gtk + 3 linux c	Martijn Pieters , Алексей Неудачин
5	Gtk3 con Ruby	arjun
6	Instalación de GTK + 3 en Windows (usando el repositorio GIT de GNOME) (GCC asumiendo el lenguaje C está instalado)	Mathews Mathai
7	Usando Glade con Builder API	Samik