# SimpleGUI User Manual

Gonzalo Hernández
Universidad de Nariño

May, 2016

## Contents

# 1 Introduction

*SimpleGUI* is an API created to be used like a base to implement Graphic User Interfaces (GUI) whereas a programmer is coding in C++, this GUI can be fabricated without to depend of any other software (Qt, GTK, etc) more than *X Window System* platform. *X Window System* is the graphic server used by unix-like operative systems to open a graphic session.

*SimpleGUI* aims to help to an programmer user when the target computer have no a special software to show graphic interfaces by mean objects like a buttons, labels, textbox, etc. All of them are designed to use the miminum resources of system, so its look is very clean.

*SimpleGUI* has the basic objects needed to create simple lightweight interfaces whereas the user is programming in C++ an over a unix-like platform.

# 2 Requirements

Due to *SimpleGUI* was implemented over *X Window System* , this API only is possible to use over Unix-like operative systems. Linux distributions are the best target to this applications, Mac is supported too by means the xQuarts project.

Is necessary to have installed the appropriate packages to use *X Window System* . Xorg is the common implementation of this graphic system, some Linux distributions require to install Xlib libraries too.

*SimpleGUI* is created over C++, so is necessary to have this development tools, g++ is the common implementation of C++.

Will be useful to have an IDE like Geany, CodeBlocks, QtCreato, etc. to programming over C++ and including the *SimpleGUI* API.

# 3 Getting Started

*SimpleGUI* is create to be easy to use, so only is necessary some simple steps to create an application.

## 3.1 Obtain the *SimpleGUI* API

*SimpleGUI* is stored in a public repository using versioning, so it can be obtained from the git repository: https://github.com/GonzaloHernandez/simplegui/releases. There are three files availables to download:

The simplegui.h is the main API, and contains all basic graphic objects needed to create an User Graphic Interface. This API is used by means the respective clause.

```
#include "simplegui.h"
```

The messagebox.h is an auxiliar API (created using simplegui.h) useful to show quick information to an user. This API can be included at top of code.

```
#include "messagebox.h"
```

The filebrowser.h is a complement applicable to search any file from the current local hard disk. Its interface show a list of file of some directory. This API must be included to be used.

```
#include "filebrowser.h"
```

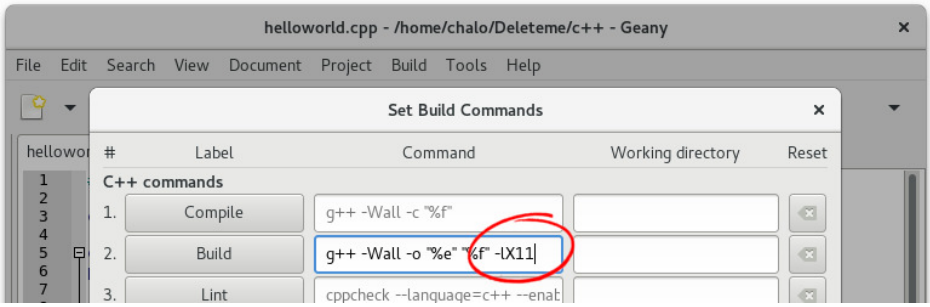
## 3.2 Compiling

To compile whatever *SimpleGUI* example, is required to pass X11 library. If the compilation is accomplished since a terminal of commands, the correct commando will be:

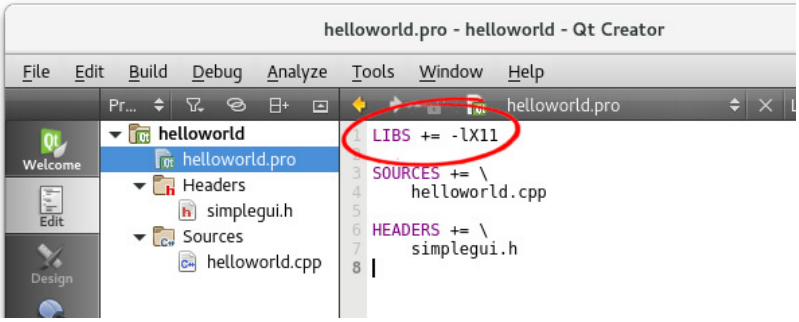
```
g++ -o helloworld helloworld.cpp -lX11
```

If is used an IDE like a Geany, QtCreator or CodeBlocks, is necessary to set some build options, the next images show each one of them.

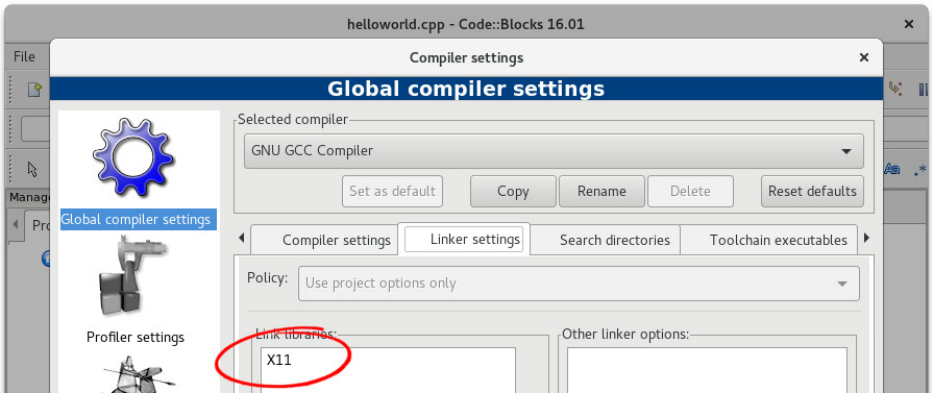
Working with Geany, the configuration is found following the menu **Build** then click in the **Set Build Commands**.




Using the QtCreator, is necessary edit the project file (*.pro).



On CodeBlocks, the option is found in the menu **settings**, next selection the option **Compiler**.




## 3.3 Creating a new *SimpleGUI* application

This secction will implement a typical "Hello World!" application by means of a guide progressive process. Is essential to follow and understand each of them before to create a more complex application.

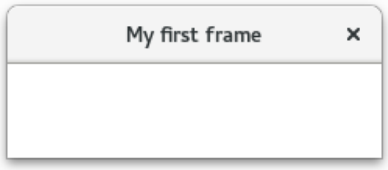
### 3.3.1 SimpleGUI Frame

To create a simple frame is needed to create a new class extended or inherited of the Frame class, the next code, show a simple example of this action.

```
#include "simplegui.h"

class HelloWorld :public Frame {
public:
    HelloWorld() : Frame(100,100,240,60,"My first frame") {
        run();
    }
};

int main() {
    HelloWorld();
}
```

**HelloWorld** class inerith all functionality of **Frame** class. When it class is instanciated, can be custom some characteristics by means of invocation of the **Frame** constructor: `Frame(x, y, width, height, title)`.

**HelloWorld** class only needs the constructor method, and is necessary to incorporate the method run() inside of this method. The main() functions create an instance of HelloWorld class. This program must be showed like a next image.




### 3.3.2 *SimpleGUI* Widgets

*SimpleGUI* offers some basic widgets to be used, in this example will be used a Button class. The next code show some new lines, to be focused en the new statemes, the code has the old lines colored with gray.

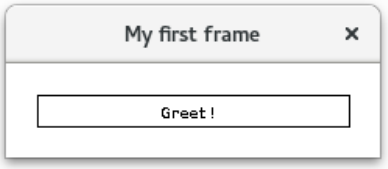
```
#include "simplegui.h"

class HelloWorld :public Frame {
private:
    Button*  greet;
public:
    HelloWorld() :  Frame(100,100,240,60,"My first frame") {
        greet = new Button(20,20,200,20,"Greet!");
        add(greet);
        run();
    }
};

int main() {
    HelloWorld();
}
```

A new element had bean programmed, the attribute **greet** is a pointer of Button class. Inside the **HelloWorld** constructor is instanciated then added to the frame. Remember to let the **run()** instruction at the end of script.

The instanciation statement and the adding statemen can be joint as such is presented in the next code.

```
        add( greet = new Button(20,20,200,20,"Greet!") );
```

After execution with these new instructions, the result must be presented such as the next image.




### 3.3.3 Event programming

Now, is time to asign a task to the button, so is necessary to insert some instructions to the program.

An event is an action recognised by the *SimpleGUI* that may be handled and programmed belatedly, so is necessary to implement a function with the wanted instructions for later be connected to the button action.

```
#include "simplegui.h"

class HelloWorld :public Frame {
private:
    Button*greet;
public:
    HelloWorld() :  Frame(100,100,240,60,"My first frame") {
        add( greet = new Button(20,20,200,20,"Greet!") );
        hello->action = &executeTask;
        run();
    }
private:
    static void executeTask() {
        exit(0);
    }
};

int main() {
    HelloWorld();
}
```

In this example, has bean created the static method `void executeTask()`, which will finish the application.

This method should tie self to the button action, the `hello->action = &executeTask` does it.

Now, when the application is running, the final user can be clicked over the button to halt the application.

### 3.3.4 Sending message to *SimpleGUI* widgets

Event Programming is handled by *SimpleGUI* through connecting static method to some widget action. Those methods must be static, then is imposible arrive to one attribute of HelloWorld from coding inside of them.

To solve this impasse, is needed to define an anchor outside of **HelloWorld** class then link these anchor with the **HelloWorld** instance address memory.

```
#include "simplegui.h"

class HelloWorld* hello;

class HelloWorld :public Frame {
private:
    Button*greet;
public:
    HelloWorld() :  Frame(100,100,240,60,"My first frame") {
        hello = this;
        add( greet = new Button(20,20,200,20,"Greet!") );
        hello->action = &executeTask;
        run();
    }
private:
```

```
    static void executeTask() {
        hello->greet->setText("Hello Word!");
    }
};

int main() {
    HelloWorld();
}
```

In this example, was created a pointer variable by means of the instruction: `class HelloWorld* hello.` This instruction must be declared before the **HelloWorld** class definition.

Afterwards this declaration, the **hello** variable will save the **HelloWorld** address memory with de instruction: `hello = this` wrote at the beginning of the **HelloWorld** constructor.

Now, the **executeTask()** method will can to show the wanted greeting ("Hello World!") in the button when the user does a click over it.

The instruction `hello->greet->setText("Hello Word!")` use the anchor **hello** to access to whatever attribute of **HelloWorld** class.

The below image show the result after do click over the button.