Easy GUI
- User Manual -

Bogdan-Nicolae David
david.bnicolae@gmail.com

March 4, 2023

Contents

1 Introduction

This document will provide you a broad overview of what this extension has to offer and how it simplifies UI creation, as well as detailed guides of how to use each major component of the extension. Do note that this document **does not** replace the documentation. If you want specific details on how to use each function, please check the documentation that comes with the release package.

As you have noticed in the **installation guide**, the extension comes with two libraries: Application and Assets. Each library deals with a different area of UI creation.

2 The Application library

This library deals with the flow control and event response of the UI. The classes provided by the library help group up visual elements into menus and easily define a flow between them. In order to create an application, you need to provide a screen size and a title for your application like so:

Figure 1: Basic application

Note: the application needs exactly **one** menu to be marked as start menu. Furthermore, the notion of *first menu* introduced in the picture referes to the **first menu to be displayed**, not the first menu to be added or created.

Now let's talk about what the functions shown above do:

- The *getInstance()* call will create an application instance with the parameters provided. Any subsequent calls will either **resize** the application (if new dimensions are provided) or return a pointer to the **same** instance created. It is worth to note that there can only be **one** application instance.
- The *addMenu()* function will add a new menu to the application. The menu can be created earlier or on the spot, depending on the arguments provided. The call will also return a pointer to the menu that has been added, so you can use it immediately after creation.
- The *start()* function will instruct the application to display and begin working. The startup process occurrs in multiple steps:

  1. The application content is checked for errors (such as duplicate menu ids or missing start menu).
  2. If no errors are found , the application window is opened and render loop is started.
  3. The application will process incoming events and render the current menu on the screen.

  In case an error occurs at this level, an **ApplicationException** will be thrown. That is why it is good practice to surround this code in a try-catch block, so that the program will not crash unexpectedly.

Now that you know how to add menus and to start our application, let's talk about how to bring some functionality into the UI via **Routines**.

2.1 Routines

A routine is a class which instructs the application how to behave when certain events occur. The application has included by default four such routines:

1. A routine for mouse clicks
2. A routine for mouse movement
3. A routine for text entered events.

The purpose of these routines is to let components in the active menu react to the specific events via handlers (e.g. onClick() and onHover()). However, sometimes that is not enough. In order to create our own routine, you need to define two key components:

- A **trigger** which will tell the routine when to fire
- A **task** which will tell the routine what to do when triggered.

In order to create a trigger, you will need to have a function which can accept an **Event** as argument and return true or false (which will tell the routine to fire), like so:

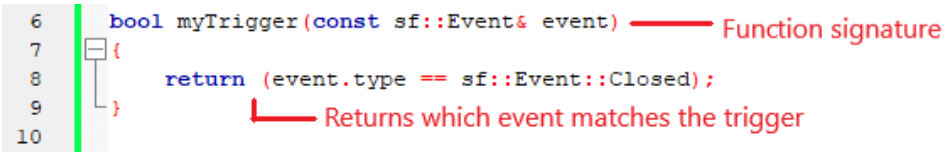
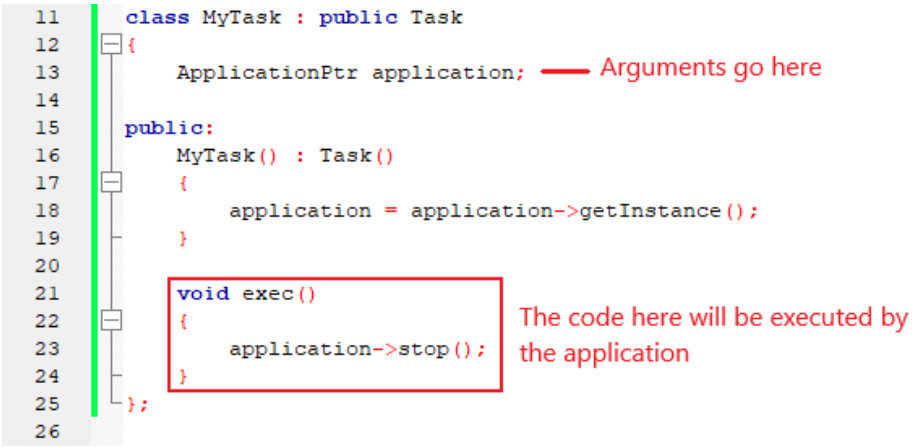


Figure 2: A routine trigger

In order to create a task, you will need to inherit a special class named **Task**. The base class provides an *exec()* method, which will be called whenever the routine will fire. A basic example has been depicted below:




```
11  class MyTask : public Task
12  {
13      ApplicationPtr application;
14
15  public:
16      MyTask() : Task()
17      {
18          application = application->getInstance();
19      }
20
21      void exec()
22      {
23          application->stop();
24      }
25  };
26
```

Figure 3: A custom task

Now that you have established what the routine will do and when it will fire, it is time to create it and add it to your application.

```
27  int main()
28  {
29      try
30      {
31          ApplicationPtr app = app->getInstance(800, 800, "Demo title");
32          Routine myRoutine(myTrigger, createNewTask(MyTask()));
33
34          app->addMenu("DemoMenu", true);
35          app->addRoutine(myRoutine);
36          app->start();
37
38          return 0;
39      }
40      catch(const ApplicationException& err)
41      {
42          ERROR << err.what();
43          return 1;
44      }
45  }
```

Figure 4: Adding routines

2.2 Menus

After you have created a menu, you can add components to it. That way, whenever the menu becomes active, the application will draw all of its components and allow them to be interacted with. Furthermore, the menu can provide access to the components to external objects (such as Routines or other menus).

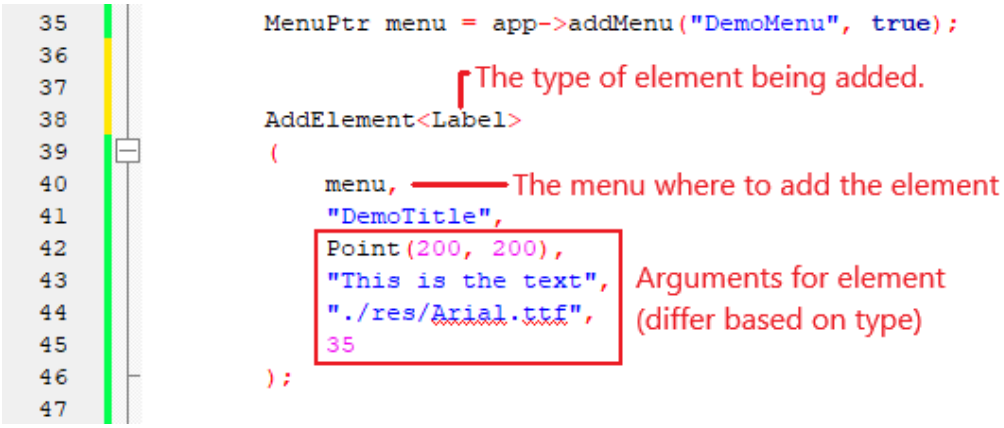


Figure 5: Adding an element

3 The Asset library

This library provides implementations for all the GUI elements EasyGUI has to offer. It works as a complement to the Application library, but can also work on its own. Each element can be interacted with via two methods:

1. *onClick()* triggers when component is being clicked
2. *onHover()* triggers when mouse enters or leaves the component area.

In order to make each component interact-able you must create a **Task** (similarly to how it is done for Routines) and add it to the component via the *setOnClick()* or *setOnHover()* methods.

For details regarding an individual component please consult the DoxyGen documentation provided with the release package.

3.1 Custom components

If you want to implement your own component, there are several methods that must implement the **draw()** method (see SFML Custom Drawable for details), as well as the **updateLocation()** method.