Chapter 1:: Introduction to Declarative Build System.

A **Graphical User Interface (GUI)** is a visual interface that allows users to interact with software through graphical elements such as buttons, labels, text fields, and windows. The **event loop** is a core mechanism that drives GUI applications. When a GUI application starts, it initializes the interface and enters a continuous loop, waiting for events like mouse clicks, key presses, or system updates. Each time an event occurs, the event loop captures it and dispatches it to the relevant part of the program, often triggering specific actions like executing a function or updating the UI. For example, when a user clicks a button, the event loop detects this click event and calls the function associated with that button (often through a callback or event handler). This event-driven model allows GUIs to be responsive, as the program doesn't run in a linear sequence but instead reacts to user inputs in real-time. The event loop keeps the application alive and responsive until the user decides to close the window, at which point the event loop ends and the application terminates.

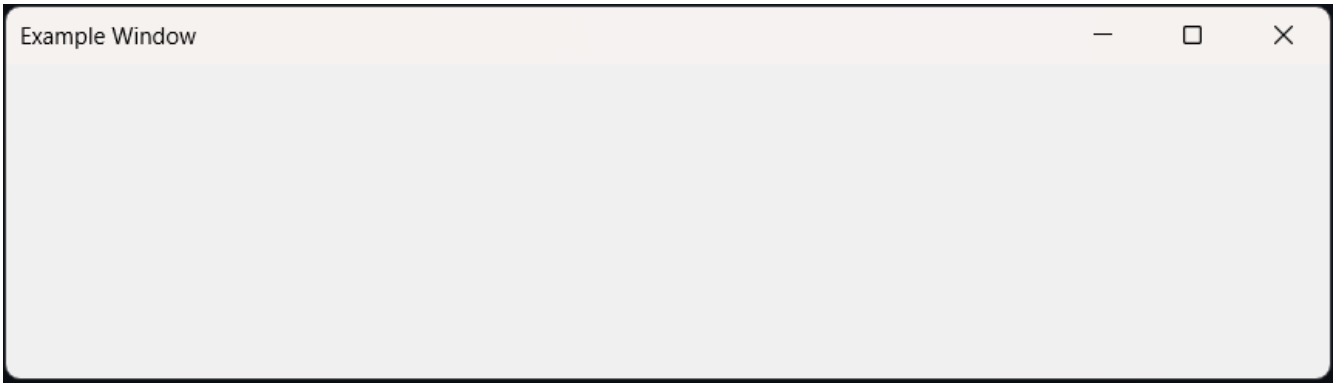
```
MyWindow() : Awindow("Sample Window", 700_dp, 200_dp)
```

In this example, an instance of `MyWindow` is created and displayed using the `show()` method. The `AUI_ENTRY` macro marks the entry point of the application, basically providing the event loop for the GUI.

```
AUI_ENTRY {
   _new<MyWindow>()->show();
   return 0;
}
```

Here is our first Window made with AUI:

```
#include <AUI/Platform/Entry.h>
#include <AUI/Platform/AWindow.h>
#include <AUI/Util/UIBuildingHelpers.h>
#include <AUI/View/ALabel.h>
#include <AUI/View/AButton.h>
#include <AUI/Platform/APlatform.h>

class MyWindow : public AWindow {
public:
    MyWindow() : AWindow("Example Window", 700_dp, 200_dp)
    {

    }
};

AUI_ENTRY{
    _new<MyWindow>()->show();

    return 0;
}
```

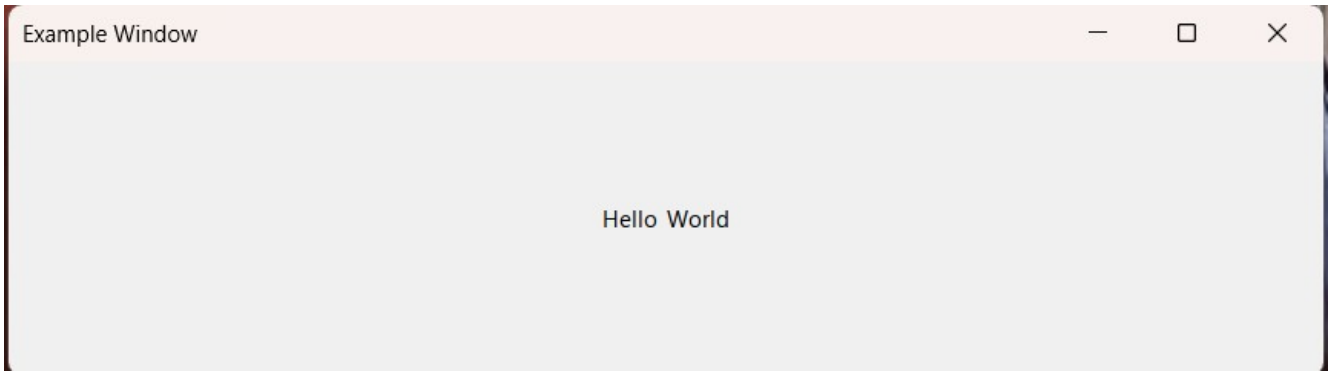


Once We have compiled everything, we will see our first AUI Application in action.

Setcontent method is used to compile UI Elements, it is basically a domain specific language embedded inside AUI Framework.

To get acess of declarative system acess them using ,

```
using namespace ass;
using namespace declarative;


class MyWindow : public AWindow {
public:
    MyWindow() : AWindow("Example Window", 700_dp, 200_dp)
    {
        setContents(
            Stacked{
                Label {"Hello World"}
            }
        );

    }
};

AUI_ENTRY{
    _new<MyWindow>()->show();

    return 0;
}
```



Once You have sucessfully compiled them, you will see the hello world label printed on your window.

Let us understand a few aspects of the Set Content GUI Builder. All elements are declarative, separated by commas, and positioning is done by the Layout Manager of the AUI VIEW system.
Let's learn how to use it by experimenting with it. As the name suggests, "Horizontal" arranges the elements in a horizontal layout, while "Vertical" arranges them vertically - simple, isn't it?

```
class MyWindow : public AWindow {
public:
    MyWindow() : AWindow("Example Window", 700_dp, 200_dp)
    {
        setContents(
            Stacked{
                Vertical{
                Label {"UP HERE"}, // , Comma separated it from element placed after it.
                Horizontal{
                Label {"1 "},
                Label {"2 "},
                Label {"3 "} // Last Element in Horizontal Declaraton
                             // Notice it is not separated by , like it's breadthren
                },
                Label {"DOWN HERE"}
                }
            }
        );

    }
};
```

Elements are placed accordingly as we exspected them to be.

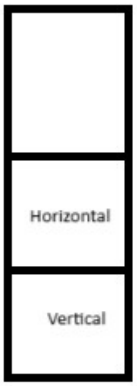
What pupose does stacked declaration serve us? It allows us to use multiple VIEW container like Horizontal and Vertical At Once.

```
class MyWindow : public AWindow {
public:
    MyWindow() : AWindow("Example Window", 700_dp, 200_dp)
    {
        setContents(

                Vertical{
                Label {"UP HERE"},
                Horizontal{
                Label {"1 "},
                Label {"2 "},
                Label {"3 "}
                },
                Label {"DOWN HERE"}
                },
               Horizontal {"Out of Vertical"}

        );

    }
};
```

The code above will not compile, as there are too many arguments for the function call. The SetContent GUI Builder can only take one layout declaration at a time; however, the stacked method stacks them for processing. The corrected code with stacked declarations will work.

Corrected Code:

```
class MyWindow : public AWindow {
public:
    MyWindow() : AWindow("Example Window", 700_dp, 200_dp)
    {
        setContents(
            Stacked{
                Vertical{
                Label {"UP HERE"},
                Horizontal{
                Label {"1 "},
                Label {"2 "},
                Label {"3 "}
                },
                Label {"DOWN HERE"}
                },
               Horizontal {"Out of Vertical"}
            }
        );

    }
};
```



Their another modifier for Layout Manager called Centred, let's see how that work before closing this chapter.

```
class MyWindow : public AWindow {
public:
    MyWindow() : AWindow("Example Window", 700_dp, 200_dp)
    {
        setContents(
            Stacked{
                Centered{ Label {"Hello World"}}
            }
        );

    }
};
```