What is SFML ?

SFML provides a simple interface to the various components of your PC, to ease the development of games and multimedia applications. It is composed of five modules: system, window, graphics, audio and network.

Simple and Fast Multimedia Library - Wikipedia

Simple and Fast Multimedia Library (SFML) is a cross-platform software development library designed to provide a simple application programming interface (API) to various multimedia

W https://en.wikipedia.org/wiki/Simple_and_Fast_Multimedia_Library

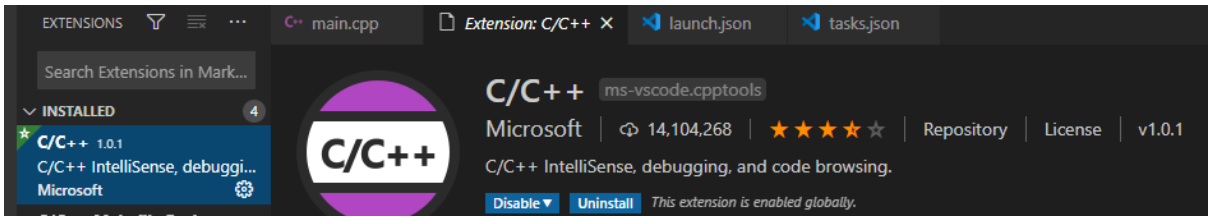
Setup SFML with Visual Studio Code and MingW on Windows

Install VS Code and MingW

The first thing that we need to install is Visual Studio Code editor. Download, install, and launch VS Code.

You need to **install the C/C++ plugin from Microsoft**

.



Now, you need to install mingw64 to use the compilers that are included like g++, if you want to use the Visual Studio compiler, you can change some of the settings here but for now we continue with mingw64.

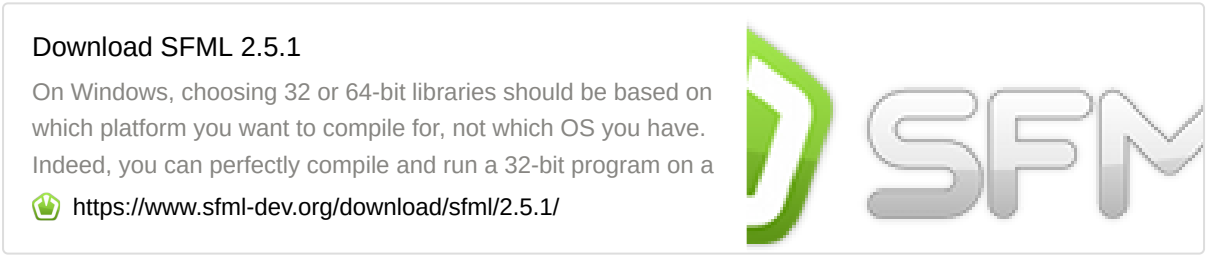
For an easier installation, use the MinGW-W64 Online Installer, take note of the path where you installed your compiler, because we're going to need it in the configuration.

Note: Be sure that you are added the bin directory of MinGW to your Windows PATH. If you don't do this step, VS Code won't be able to find the proper tools!

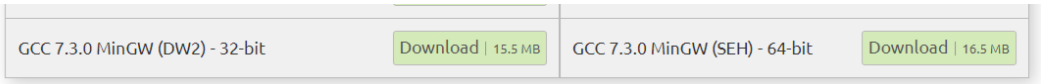
Now, we need to download **SFML** For our case, we have to download the Development libraries for MinGW.

Install SFML

- Download the SFML libraries for windows.



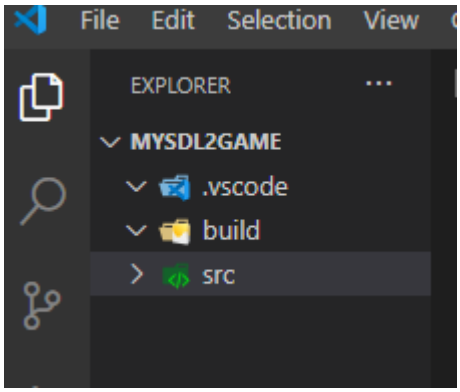
- Choose the 32bit or 64bit according to your computer architecture.



- Extract the downloaded zip-file and paste it into C:\ disk.

Creating the SFML project

Now we are ready to start our SFML project. Create a folder for your project and open it with **VSCode** . Inside the folder, create another called **src**, here is where are going to save our code files. For our SFML and C++ configuration, create a **.vscode** folder. For the output of our compilation, create a **build** folder. Your editor should be like this:



The first thing that we need to create is our Task to compile our code. In the **.vscode** folder, create a new file and name it **tasks.json**. Inside the file, write the following

configuration.

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "shell",
      "label": "SFML",
      "command": "C:\\mingw64\\bin\\g++.exe",
      "args": [
        "-g",
        "src\\*.cpp",
        "-o",
        "build\\game.exe",
        "-IC:\\SFML-2.5.1\\include",
        "-LC:\\SFML-2.5.1\\lib",
        "-lsfml-graphics",
        "-lsfml-window",
        "-lsfml-system",
        "-lopengl32",
        "-lfreetype",
        "-lwinmm",
        "-lgdi32",
        "-mwindows",
        "-lsfml-main"
      ],
      "options": {
        "cwd": "${workspaceFolder}"
      },
      "problemMatcher": ["$gcc"],
      "group": {
        "kind": "build",
        "isDefault": true
      }
    },
    {
      "type": "cppbuild",
      "label": "C/C++: cl.exe build active file",
      "command": "cl.exe",
      "args": [
        "/Zi",
        "/EHsc",
        "/nologo",
        "/Fe:",
        "${fileDirname}\\${fileBasenameNoExtension}.exe",
        "${file}"
      ],
      "options": {
        "cwd": "${fileDirname}"
      },
      "problemMatcher": ["$msCompile"],
      "group": "build",
      "detail": "compiler: cl.exe"
    }
  ]
}
```

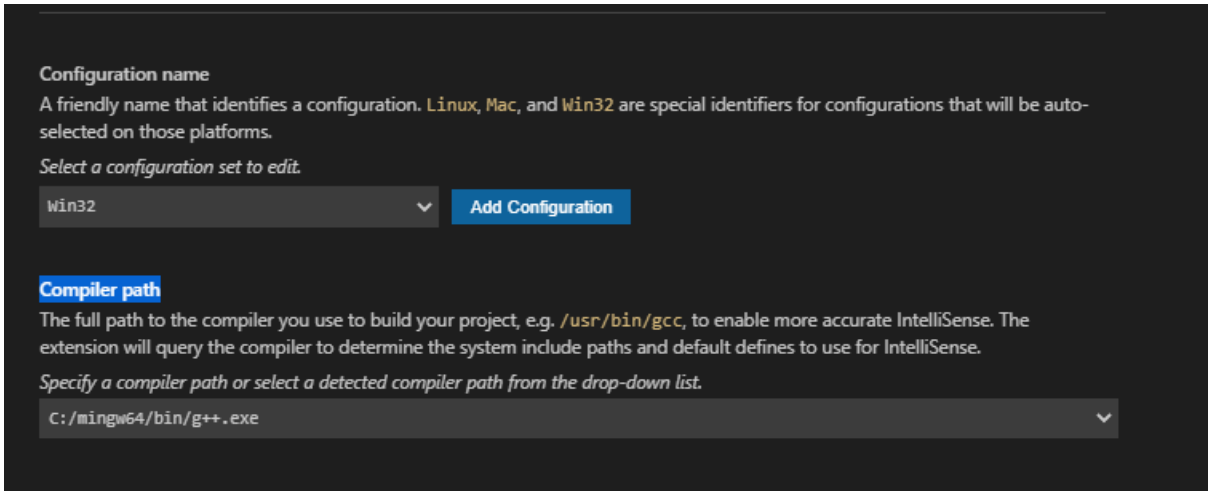
Now, we need to create a **launch.json** file to launch our game and debugging.

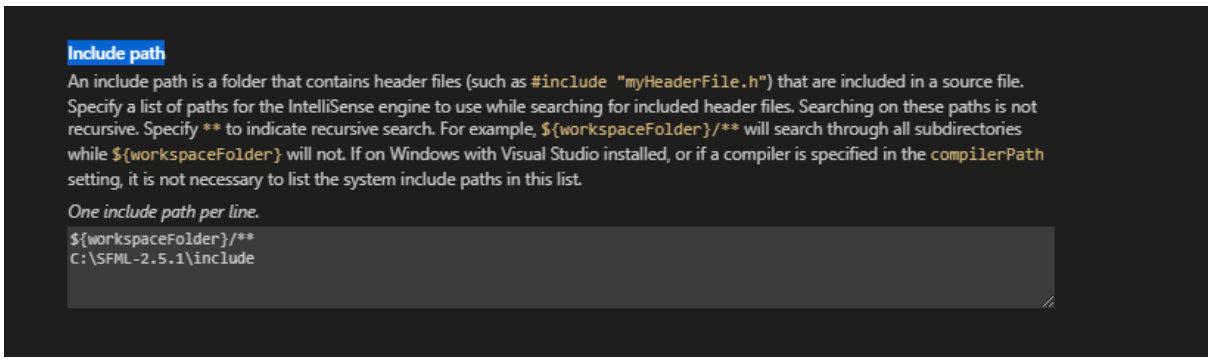
```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "(gdb)",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}\\build\\.exe",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "miDebuggerPath": "C:\\mingw64\\bin\\gdb.exe",
            "setupCommands": [
                {
                    "description": "Enably pretty printing",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "SFML"
        }
    ]
}
```

Now, as the last configuration step, we need to configure our C++ options to have a better Intellisense autocomplete and configuration of our compiler.

In Visual Studio Code, press **CTRL + SHIFT + P**, write C/C++ and select **C/C++ Edit configurations (GUI)**. Here we’re are going to change some configurations.

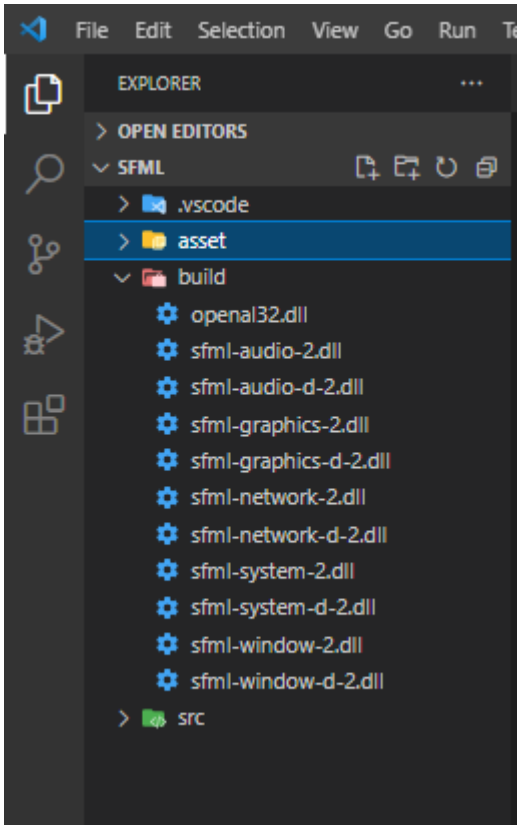
In **Compiler Path**, you need to put the path to your compiler. Like we put in the tasks.json and change the IntelliSense mode to gcc_x64. In Include Path, add the path to your SFML includes.





Other options that you may change are **C Standard** and **C++ Standard**, If you save these configurations, you are going to have a file named **c_cpp_properties.json** in your .vscode folder.

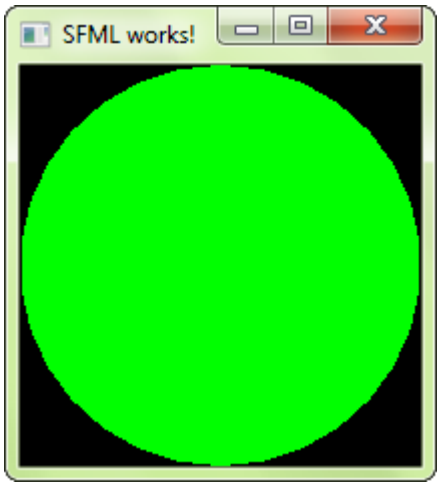
Now, you need to put the **dll** file in the **build** folder, to be able to run our SFML game. you are going to find the dll files here **C:\SFML-2.5.1\bin**



Testing our SFML game

```
#include <SFML/Graphics.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(200, 200), "SFML works!");
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();
        window.draw(shape);
        window.display();
    }

    return 0;
}
```

Press **Ctrl + Shift + B** and go to build folder and see if game.exe file is created or not if the file is created that means your setup is sucessful. Now run the game.exe file and you will see this.



This is a simple configuration to start with SFML.