LEARNING vscode

#vscode

Table of Contents

About

Chapter 1: Getting started with vscode

Remarks

This section provides an overview of what vscode is, and why a developer might want to use it.

It should also mention any large subjects within vscode, and link out to the related topics. Since the Documentation for vscode is new, you may need to create initial versions of those related topics.

Versions

Version	Release date
0.10.1-extensionbuilders	2015-11-13
0.10.1	2015-11-17
0.10.2	2015-11-24
0.10.3	2015-11-26
0.10.5	2015-12-17
0.10.6	2015-12-19
0.10.7-insiders	2016-01-29
0.10.8	2016-02-05
0.10.8-insiders	2016-02-08
0.10.9	2016-02-17
0.10.10-insiders	2016-02-26
0.10.10	2016-03-11
0.10.11	2016-03-11
0.10.11-insiders	2016-03-11
0.10.12-insiders	2016-03-20
0.10.13-insiders	2016-03-29
0.10.14-insiders	2016-04-04

Version	Release date
0.10.15-insiders	2016-04-11
1.0.0	**2016-04-14**
1.1.0-insider	2016-05-02
1.1.0	2016-05-09
1.1.1	2016-05-16
1.2.0	2016-06-01
1.2.1	2016-06-14
1.3.0	2016-07-07
1.3.1	2016-07-12
1.4.0	2016-08-03
translation/20160817.01	2016-08-17
translation/20160826.01	2016-08-26
translation/20160902.01	2016-09-02
1.5.0	2016-09-08
1.5.1	2016-09-08
1.5.2	2016-09-14
1.6.0	2016-10-10
1.6.1	2016-10-13
translation/20161014.01	2016-10-14
translation/20161028.01	2016-10-28
1.7.0	2016-11-01
1.7.1	2016-11-03
translation/20161111.01	2016-11-12
translation/20161118.01	2016-11-19
1.7.2	2016-11-22

Version	Release date
translation/20161125.01	2016-11-26
translation/20161209.01	2016-12-09
1.8.0	2016-12-14
1.8.1	2016-12-20
translation/20170123.01	2017-01-23
translation/20172701.01	2017-01-27
1.9.0	2017-02-02
translation/20170127.01	2017-02-03
translation/20170203.01	2017-02-03
1.9.1	2017-02-09
translation/20170217.01	2017-02-18
translation/20170227.01	2017-02-27
1.10.0	2017-03-01
1.10.1	2017-03-02
1.10.2	2017-03-08
translation/20170311.01	2017-03-11
translation/20170317.01	2017-03-18
translation/20170324.01	2017-03-25
translation/20170331.01	2017-03-31
1.11.0	2017-04-06
1.11.1	2017-04-06
translation/20170407.01	2017-04-07
1.11.2	2017-04-13

Examples

On Windows

- Download the Visual Studio Code installer for Windows.
- Once it is downloaded, run the installer (VSCodeSetup-version.exe). This will only take a minute.

By default, VS Code is installed under C:\Program Files (x86)\Microsoft VS Code for a 64-bit machine.

Note: .NET Framework 4.5.2 is required for VS Code. If you are using Windows 7, please make sure .NET Framework 4.5.2 is installed.

Tip: Setup will optionally add Visual Studio Code to your %PATH%, so from the console you can type 'code .' to open VS Code on that folder. You will need to restart your console after the installation for the change to the %PATH% environmental variable to take effect.

On Mac

- Download Visual Studio Code for Mac.
- Double-click on the downloaded archive to expand the contents.
- Drag Visual Studio Code.app to the Applications folder, making it available in the Launchpad.
- Add VS Code to your Dock by right-clicking on the icon and choosing Options, Keep in Dock.

You can also run VS Code from the terminal by typing 'code' after adding it to the path:

- Launch VS Code.
- Open the Command Palette (Ctrl+Shift+P) and type 'shell command' to find the Shell Command: Install 'code' command in PATH command.

Restart the terminal for the new $PATH value to take effect. You'll be able to type 'code .' in any folder to start editing files in that folder.

Note: If you still have the old code alias in your .bash_profile (or equivalent) from an early VS Code version, remove it and replace it by executing the Shell Command: Install 'code' command in PATH command.

On Linux

Debian and Ubuntu based distributions

The easiest way to install for Debian/Ubuntu based distributions is to download and install the .deb package (64-bit) either through the graphical software center if it's available or through the command line with:

```
sudo dpkg -i <file>.deb
sudo apt-get install -f # Install dependencies
```

Installing the .deb package will automatically install the apt repository and signing key to enable auto-updating using the regular system mechanism. Note that 32-bit and .tar.gz binaries are also available on the download page.

The repository and key can also be installed manually with the following script:

```
curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.gpg
sudo mv microsoft.gpg /etc/apt/trusted.gpg.d/microsoft.gpg
sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/vscode stable main" >
/etc/apt/sources.list.d/vscode.list'
```

Then update the package cache and install the package using:

```
sudo apt-get update
sudo apt-get install code # or code-insiders for insiders build
```

RHEL, Fedora and CentOS based distributions

We currently ship the stable 64-bit VS Code in a yum repository, the following script will install the key and repository:

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
sudo sh -c 'echo -e "[code]\nname=Visual Studio
Code\nbaseurl=https://packages.microsoft.com/yumrepos/vscode\nenabled=1\ngpgcheck=1\ngpgkey=https://pac
> /etc/yum.repos.d/vscode.repo'
```

Then update the package cache and install the package using dnf (Fedora 22 and above):

```
dnf check-update
sudo dnf install code
```

Or on older versions using yum:

```
yum check-update
sudo yum install code
```

openSUSE and SLE based distributions

The yum repository above also works for openSUSE and SLE based systems, the following script will install the key and repository:

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
sudo sh -c 'echo -e "[code]\nname=Visual Studio
Code\nbaseurl=https://packages.microsoft.com/yumrepos/vscode\nenabled=1\ntype=rpm-
md\ngpgcheck=1\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" >
/etc/zypp/repos.d/vscode.repo'
```

Then update the package cache and install the package using:

```
sudo zypper refresh
sudo zypper install code
```

AUR package for Arch Linux

There is a community maintained Arch User Repository (AUR) package for VS Code.

Installing .rpm package manually The .rpm package (64-bit) can also be manually downloaded and installed, however auto-updating won't work unless the repository above is installed. Once downloaded it can be installed using your package manager, for example with dnf:

```
sudo dnf install <file>.rpm
```

Note that 32-bit and .tar.gz binaries are are also available on the download page.

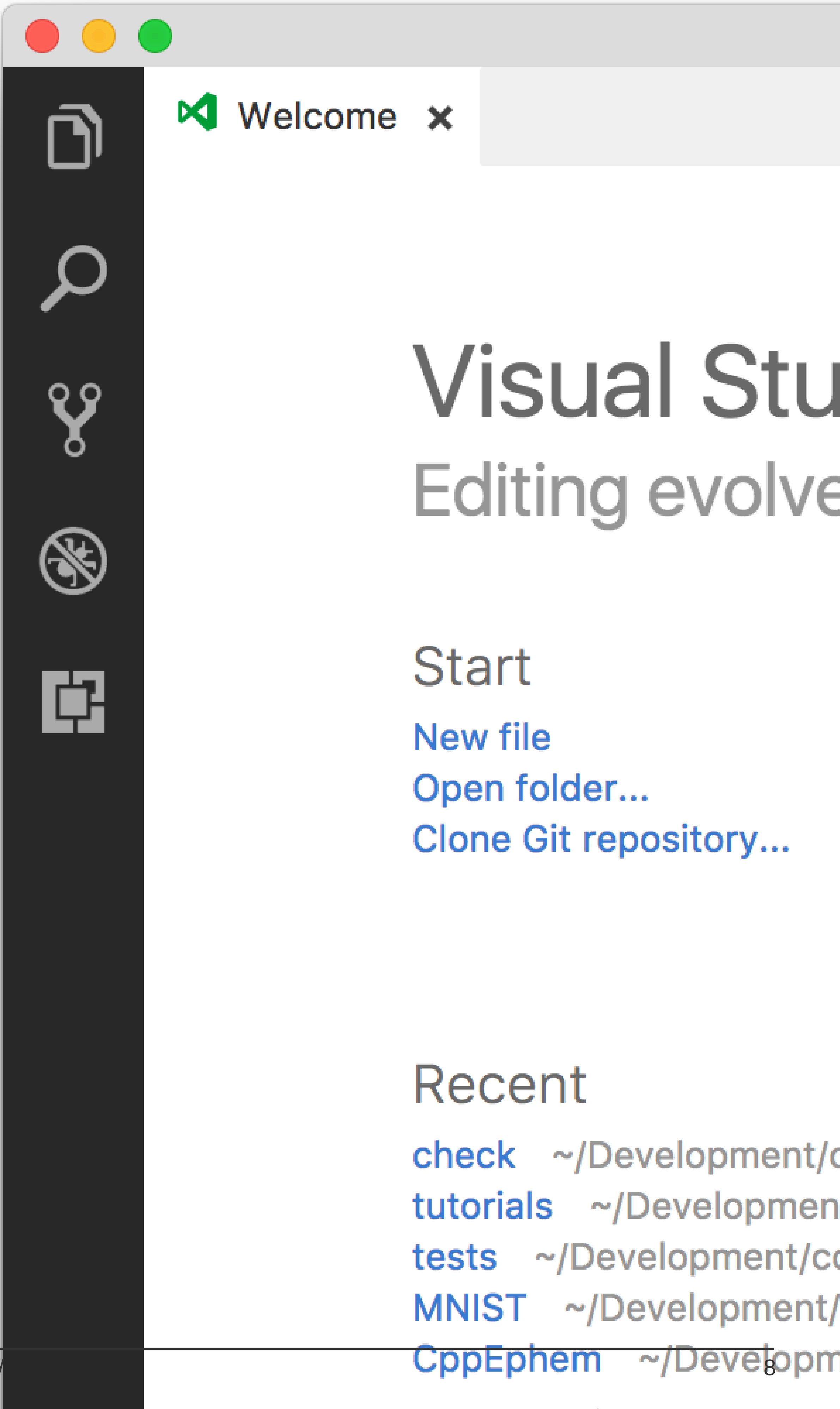
First Steps (C++): HelloWorld.cpp

The first program one typically writes in any language is the "hello world" script. This example demonstrates how to write this program and debug it using Visual Studio Code (I'll refer to Visual Studio Code as VS Code from now on).

Create The Project

Step 1 will be to create a new project. This can be done in a number of ways. The first way is directly from the user interface.

1. Open VS Code program. You will be greeted with the standard welcome screen (note the images are taken while working on a Mac, but they should be similar to your installation):


Welcome ×
Visual Stu
Editing evolve
Start
New file
Open folder...
Clone Git repository...
Recent
check ~/Development/c
tutorials ~/Developmen
tests ~/Development/c
MNIST ~/Development/
CppEphem ~/Developm

. This will open a new editing window where we can begin constructing our script. Go ahead and save this file (you can use the menu *File* > *Save* to do this). For this example we will call the file **HelloWorld.cpp** and place it in a new directory which we will call **VSC_HelloWorld/**.

3. Write the program. This should be fairly straight forward, but feel free to copy the following into the file:

```
#include <iostream>

int main() {
   std::cout << "Hello world!" << std::endl;
   return 0;
}
```

Run the Code

Next we want to run the script and check its output. There are a number of ways to do this. The simplest is to open a terminal, and navigate to the directory that we created. You can now compile the script and run it with gcc by typing:

```
$ g++ HelloWorld.cpp -o helloworld
$ ./helloworld
Hello World!
```

Yay, the program worked! But this isn't really what we want. It would be much better if we could run the program from within VSCode itself. We're in luck though! VSCode has a built in terminal which we can access via the menu "**View**" > "**Integrated Terminal**". This will open a terminal in the bottom half of the window from which you can navigate to the **VSC_HelloWorld** directory and run the above commands.

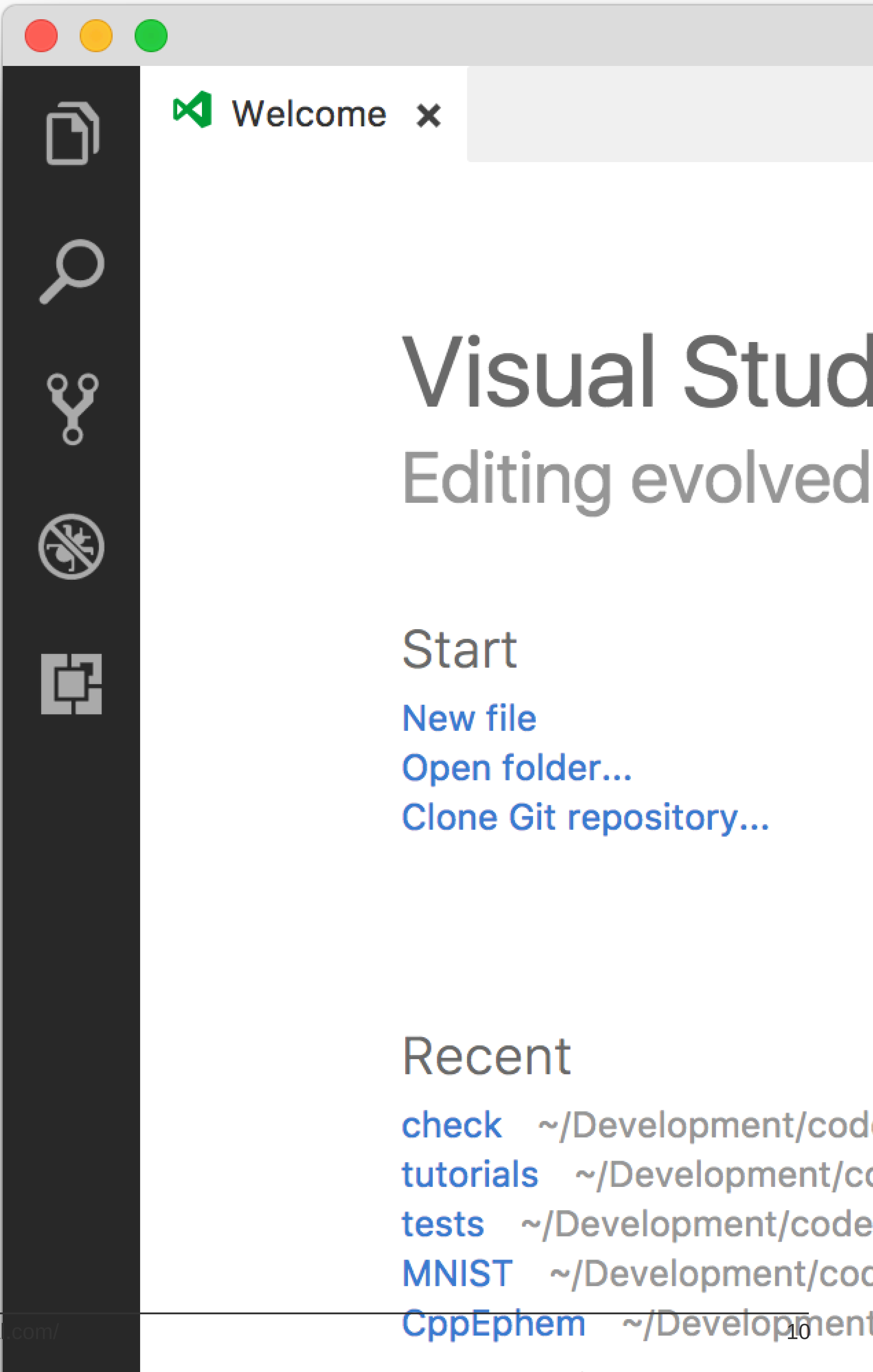
Typically we do this by executing a *Run Task*. From the menu select "**Tasks**" > "**Run Task...**". You'll notice you get a small popup near the top of the window with an error message (something along the lines of

First program (C++): Hello World.cpp

This example introduces you to the basic functionality of VS Code by demonstrating how to write a "hello world" program in C++. Before continuing, make sure you have the "**ms-vscode.cpptools**" extension installed.

Initialize the Project

The first step is to create a new project. To do this, load the VS Code program. You should be greeted with the typical welcome screen:


Welcome ×

Visual Stud

Editing evolved

Start

New file
Open folder...
Clone Git repository...

Recent

check ~/Development/cod
tutorials ~/Development/c
tests ~/Development/code
MNIST ~/Development/coc
CppEphem ~/Development

" from the welcome screen. This will open a new file window. Go ahead and save the file ("**File**" > "**Save**") into a new directory. You can name the directory anything you want, but this example will call the directory "**VSC_HelloWorld**" and the file "**HelloWorld.cpp**".

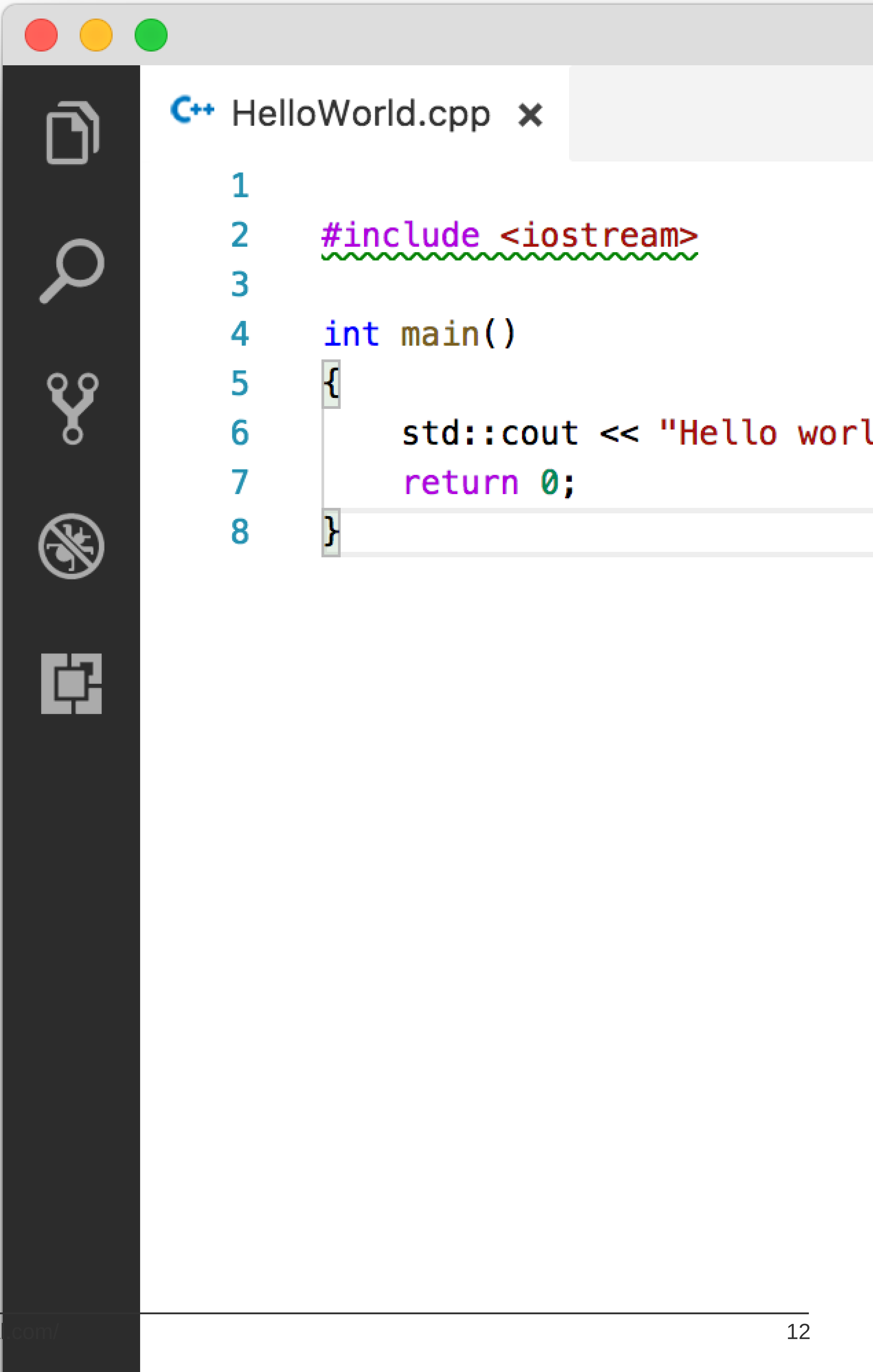
Now write the actual program (feel free to copy the below text):

```
#include <iostream>

int main()
{
    // Output the hello world text
    std::cout << "Hello world!" << std::endl;
    return 0;
}
```

Great! You'll also notice that because you've installed the "**ms-vscode.cpptools**" extension you also have pretty code-highlighting. Now let's move on to running the code.

Running the Script (basic)

We can run "**HelloWorld.cpp**" from within VS Code itself. The simplest way to run such a program is to open the integrated terminal ("**View**" > "**Integrated Terminal**"). This opens a terminal window in the lower portion of the view. From inside this terminal we can navigate to our created directory, build, and execute the script we've written.



```

#include <iostream>

int main()
{
    std::cout << "Hello worl
    return 0;
}
```

Here we've used the following commands to compile and run the code:

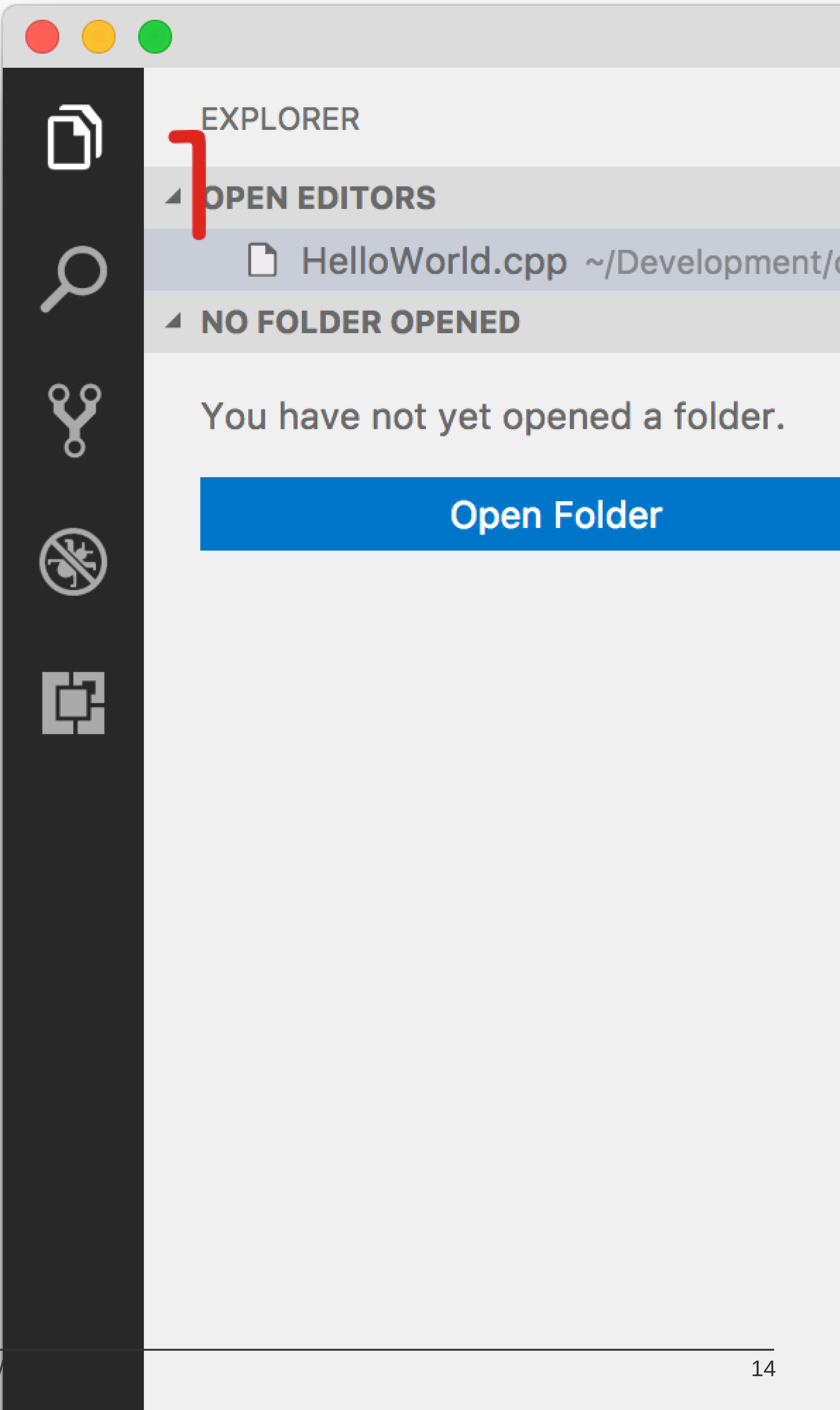
```
$ g++ HelloWorld.cpp -o hellowold
$ ./hellowold
```

Notice that we get the expected `Hello World!` output.

Running the Script (slightly more advanced)

Great, but we can use VS Code directly to build and execute the code as well. For this, we first need to turn the "**VSC_HelloWorld**" directory into a workspace. This can be done by:

1. Opening the *Explorer* menu (top most item on the vertical menu on the far left)
2. Select the *Open Folder* button
3. Select the "**VSC_HelloWorld**" directory we've been working in.



You have not yet opened a folder.

> *Note: If you open a directory within VS Code (using "**File**" > "**Open...**" for example) you will already be in a workspace.*

The *Explorer* menu now displays the contents of the directory.

Next we want to define the actual tasks which we want VS Code to run. To do this, select "**Tasks**" > "**Configure Default Build Task**". In the drop down menu, select "**Other**". This opens a new file called "**tasks.json**" which contains some default values for a task. We need to change these values. Update this file to contain the following and save it:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "taskName": "build",
            "type": "shell",
            "command": "g++ HelloWorld.cpp -o helloworld"
        },
        {
            "taskName": "run",
            "type": "shell",
            "command": "${workspaceRoot}/helloworld"
        }
    ]
}
```

*Note that the above also creates a hidden **.vscode** directory within our working directory. This is where VS Code puts configuration files including any project specific settings files. You can find out more about Tasks here.*

In the above example, `${workspaceRoot}` references the top level directory of our workspace, which is our "**VSC_HelloWorld**" directory. Now, to build the project from inside the method select "**Tasks** " > "**Run Build Task...**" and select our created "**build**" task and "*Continue without scanning the task output*" from the drop down menus that show up. Then we can run the executable using " **Tasks**" > "**Run Task...**" and selecting the "**run**" task we created. If you have the integrated terminal open, you'll notice that the "Hello World!" text will be printed there.

It is possible that the terminal may close before you are able to view the output. If this happens you can insert a line of code like this `int i; std::cin >> i;` just before the return statement at the end of the `main()` function. You can then end the script by typing any number and pressing *<Enter>*.

And that's it! You can now start writing and running your C++ scripts from within VS Code.

Read Getting started with vscode online: https://riptutorial.com/vscode/topic/4489/getting-started-with-vscode

Chapter 2: Multiple projects set up

Remarks

Unit tests project set up currently can be found here

Examples

Referencing local projects

There is no such things as `.sln` and `.proj` files.
Instead of them **folders** are being used in Visual Studio Code.
Each project folder should have a seperate `project.json` file.

```
/MyProject.Core
   SourceFile.cs
   project.json

/MyProject.Web
   /Controllers
   /Views
   project.json
```

To reference `MyProject.Core` from `MyProject.Web` project edit `MyProject.Web\project.json` file and add the dependency:

```
// MyProject.Web/project.json
{
   "dependencies": {
      "MyProject.Core": {"target": "project"},
    ...
   }
   "buildOptions": {
      "emitEntryPoint": true
   }
}
```

The line `"emitEntryPoint": true` says that `MyProject.Web` is a start project for the solution.
`MyProject.Core` should have this flag disabled in its `project.json` file:

```
 // MyProject.Core/project.json
 {
   ...
   "buildOptions": {
      "emitEntryPoint": false
   }
}
```

Build the project (Mac: ⌘+`Shift`+`B`, Windows: `Ctrl`+`Shift`+`B`) and each project should have own `\bin` and `\obj` folders with new `.dll` files.

Solution structure

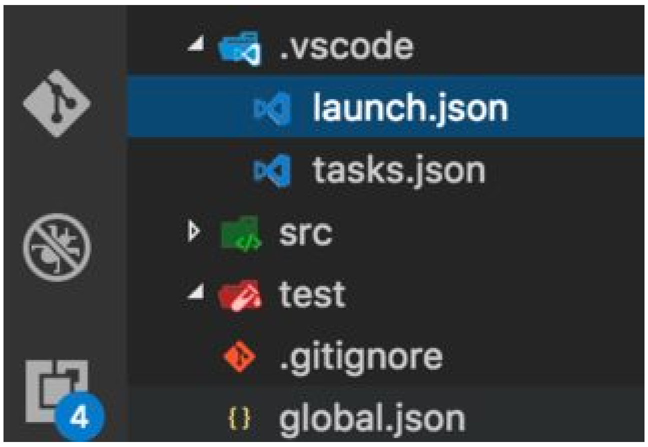
It is very common to group projects, for example, place test projects under the `/test` folder and source projects under the `/src` folder. Add `global.json` file and make similar structure:

```
global.json
/src/
    /MyProject.Core/
       SourceFile.cs
       project.json

    /MyProject.Web/
       /Controllers
       /Views
       project.json

/test/
    /MyProject.Core.UnitTests/
       SourceFileTest.cs
       project.json

    /MyProject.Web.UnitTests/
       /Controllers
       /Views
       project.json
```

Edit empty `global.json` file and specify project groups:

```
{
    "projects":["src", "test"]
}
```

VS Code uses `tasks.json` to run tasks (e.g. building a solution) and `launch.json` for starting a project (e.g. debugging). If you cannot find these files try to start debugging by pressing `F5` and ignore errors, VS Code will generate under the root folder `.vscode` folder with the files.



Edit `launch.json` file and specify the path to your start up library, change `MyProject.Web` with your project name:

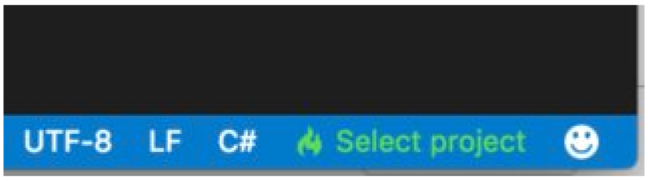
```
{
    "configurations": [
    {
        ...
        "program":
"${workspaceRoot}/src/MyProject.Web/bin/Debug/netcoreapp1.0/MyProject.Web.dll",
        "args": [],
        "cwd": "${workspaceRoot}/src/Washita.Web",
        ...
   }
}
```

Edit `tasks.json` file and specify the path to your start up library, change `MyProject.Web` with your project name:

```
{
    "tasks": [
        {
        "taskName": "build",
            "args": [
                "${workspaceRoot}/src/MyProject.Web"
            ],
            "isBuildCommand": true,
            "problemMatcher": "$msCompile"
       }
   ]
}
```

Now you should be able to build and debug .NET source files.

However Intellisense will disappear due the multiple project configuration. To fix it open any `.cs` file and switch to the appropriate project (project.json) by choosing `Select project` in the bottom right corner:



Read Multiple projects set up online: https://riptutorial.com/vscode/topic/7717/multiple-projects-set-up

Credits

S. No	Chapters	Contributors
1	Getting started with vscode	Community, H. Pauwelyn, Jvinniec, Kronos, RamenChef, Sahan Serasinghe
2	Multiple projects set up	Artru