


Topics In this article  (<https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/cpp/config-clang-mac.md>)

## Using Clang in Visual Studio Code

In this tutorial, you configure Visual Studio Code on macOS to use the Clang/LLVM compiler and debugger.

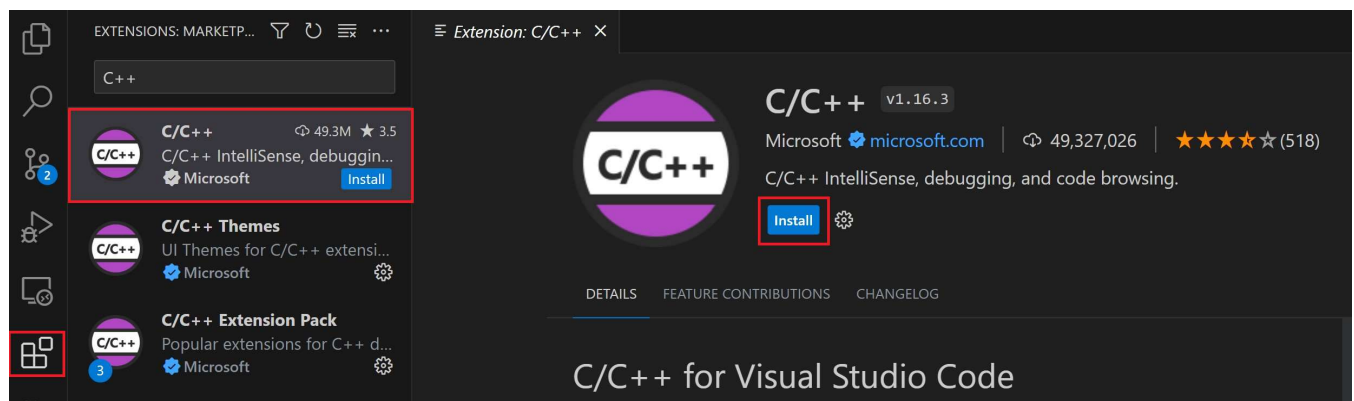
After configuring VS Code, you will compile and debug a simple C++ program in VS Code. This tutorial does not teach you about Clang or the C++ language. For those subjects, there are many good resources available on the Web.

If you have any trouble, feel free to file an issue for this tutorial in the VS Code documentation repository (<https://github.com/microsoft/vscode-docs/issues>).

### Prerequisites

To successfully complete this tutorial, you must do the following:

1. Install Visual Studio Code on macOS (</docs/setup/mac>).
2. Install the C++ extension for VS Code (<https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools>). You can install the C/C++ extension by searching for 'c++' in the Extensions view (Ctrl+Shift+X).



Ensure Clang is installed

Clang may already be installed on your Mac. To verify that it is, open a macOS Terminal window and enter the following command:

```
clang --version
```

1. If Clang isn't installed, enter the following command to install the command line developer tools:

```
xcode-select --install
```

## Create Hello World

From the macOS Terminal, create an empty folder called `projects` where you can store all your VS Code projects, then create a subfolder called `helloworld`, navigate into it, and open VS Code in that folder by entering the following commands:

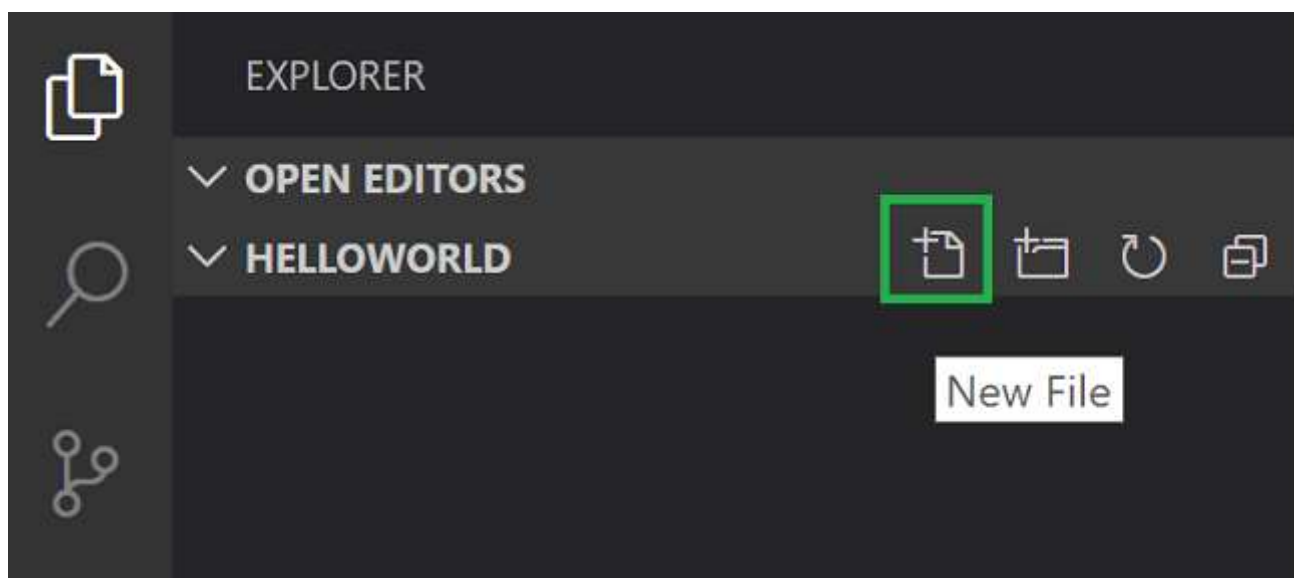
```
mkdir projects  
cd projects  
mkdir helloworld  
cd helloworld  
code .
```

The `code .` command opens VS Code in the current working folder, which becomes your "workspace". As you go through the tutorial, you will create three files in a `.vscode` folder in the workspace:

- `tasks.json` (compiler build settings)
- `launch.json` (debugger settings)
- `c_cpp_properties.json` (compiler path and IntelliSense settings)

Add hello world source code file

In the File Explorer title bar, select **New File** and name the file `helloworld.cpp`.



Paste in the following source code:

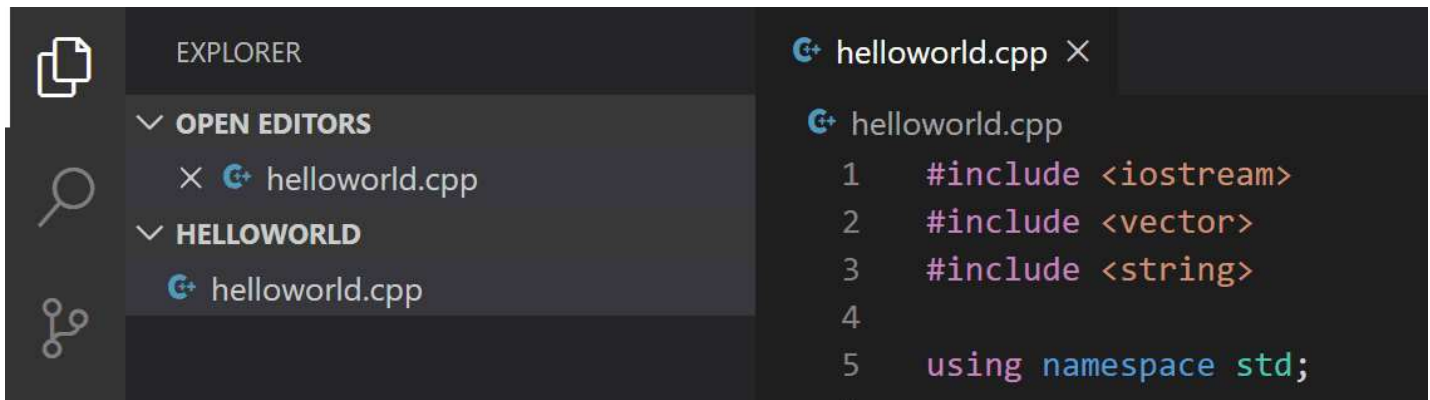
```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main()
{
    vector<string> msg {"Hello", "C++", "World", "from", "VS Code", "and the C++ extension!"};

    for (const string& word : msg)
    {
        cout << word << " ";
    }
    cout << endl;
}
```

Now press **Ctrl+S** to save the file. Notice that your files are listed in the **File Explorer** view ( **Ctrl+Shift+E** ) in the side bar of VS Code:



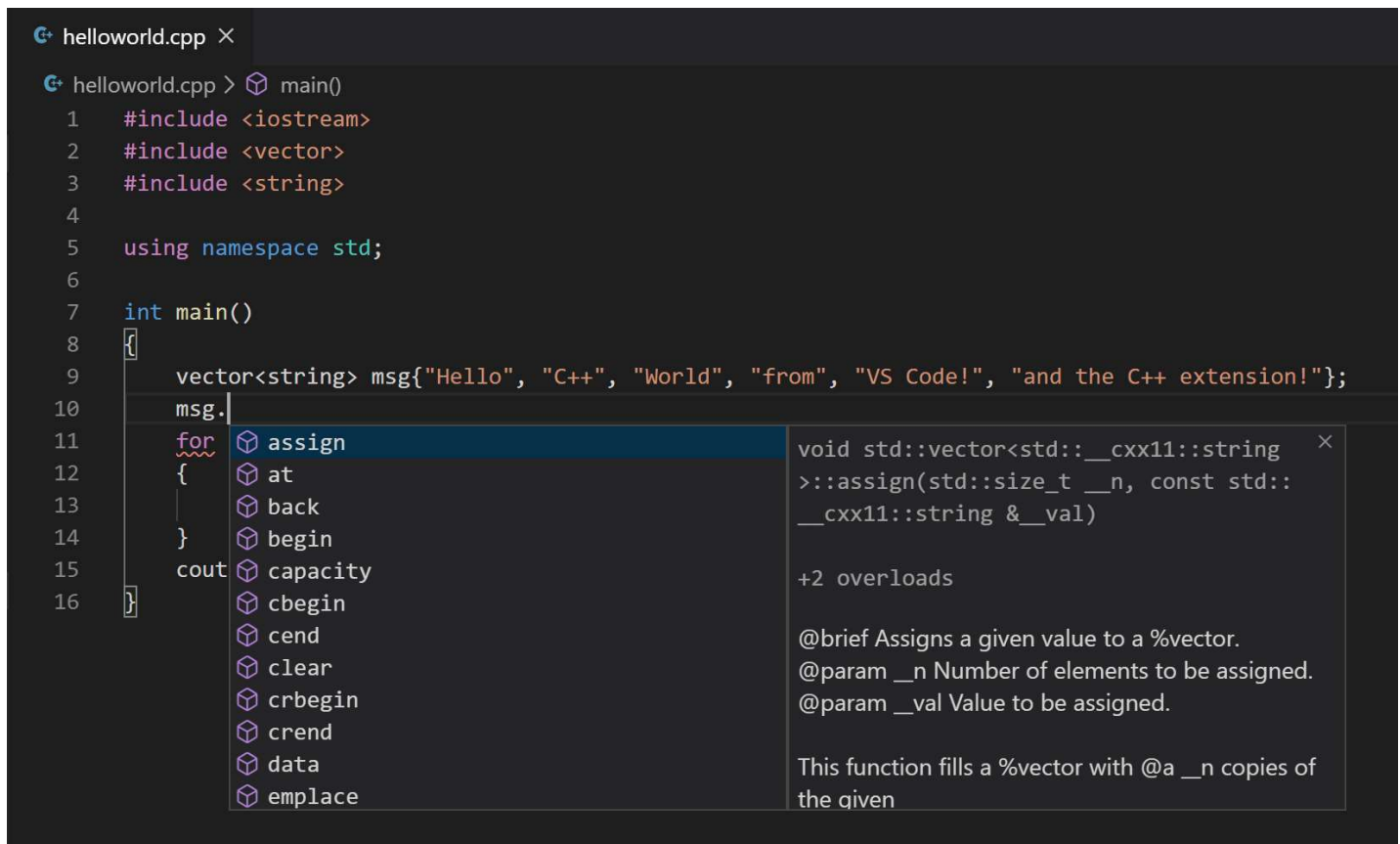
You can also enable Auto Save ([/docs/editor/codebasics#\\_save-auto-save](/docs/editor/codebasics#_save-auto-save)) to automatically save your file changes, by checking **Auto Save** in the main **File** menu.

The Activity Bar on the edge of Visual Studio Code lets you open different views such as **Search**, **Source Control**, and **Run**. You'll look at the **Run** view later in this tutorial. You can find out more about the other views in the VS Code User Interface documentation (</docs/getstarted/userinterface>).

**Note:** When you save or open a C++ file, you may see a notification from the C/C++ extension about the availability of an Insiders version, which lets you test new features and fixes. You can ignore this notification by selecting the **x** (**Clear Notification**).

## Explore IntelliSense

In the `helloworld.cpp` file, hover over `vector` or `string` to see type information. After the declaration of the `msg` variable, start typing `msg.` as you would when calling a member function. You should immediately see a completion list that shows all the member functions, and a window that shows the type information for the `msg` object:



You can press the `Tab` key to insert the selected member. Then, when you add the opening parenthesis, you'll see information about arguments that the function requires.

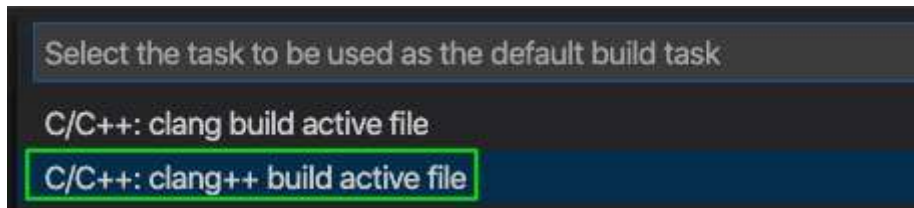
## Run helloworld.cpp

Remember, the C++ extension uses the C++ compiler you have installed on your machine to build your program. Make sure you have a C++ compiler installed before attempting to run and debug `helloworld.cpp` in VS Code.

1. Open `helloworld.cpp` so that it is the active file.
2. Press the play button in the top right corner of the editor.

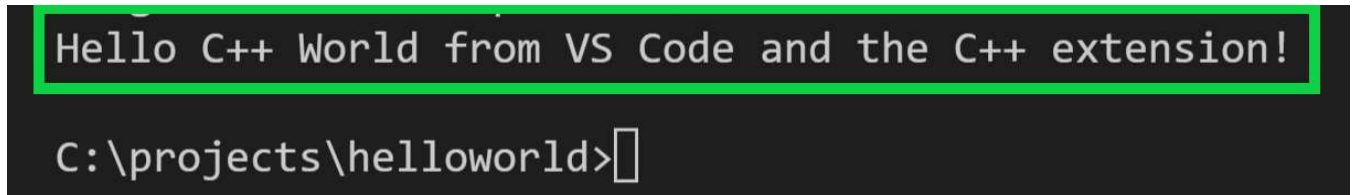


3. Choose **C/C++: clang++ build and debug active file** from the list of detected compilers on your system.



You'll only be asked to choose a compiler the first time you run `helloworld.cpp`. This compiler will be set as the "default" compiler in `tasks.json` file.

4. After the build succeeds, your program's output will appear in the integrated **Terminal**.



The first time you run your program, the C++ extension creates `tasks.json`, which you'll find in your project's `.vscode` folder. `tasks.json` stores build configurations.

Your new `tasks.json` file should look similar to the JSON below:

```
{
  // See https://go.microsoft.com/fwlink/?LinkId=733558
  // for the documentation about the tasks.json format
  "version": "2.0.0",
  "tasks": [
    {
      "type": "shell",
      "label": "C/C++: clang++ build active file",
      "command": "/usr/bin/clang++",
      "args": [
        "-std=c++17",
        "-stdlib=libc++",
        "-g",
        "${file}",
        "-o",
        "${fileDirname}/${fileBasenameNoExtension}"
      ],
      "options": {
        "cwd": "${workspaceFolder}"
      },
      "problemMatcher": ["$gcc"],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "detail": "Task generated by Debugger."
    }
  ]
}
```

**Note:** You can learn more about `tasks.json` variables in the variables reference (/docs/editor/variables-reference).

The `command` setting specifies the program to run. In this case, `"clang++"` is the driver that causes the Clang compiler to expect C++ code and link against the C++ standard library.

The `args` array specifies the command-line arguments that will be passed to `clang++`. These arguments must be specified in the order expected by the compiler.

This task tells the C++ compiler to compile the active file ( `${file}` ), and create an output file ( `-o` switch) in the current directory ( `${fileDirname}` ) with the same name as the active file ( `${fileBasenameNoExtension}` ), resulting in `helloworld` for our example.

The `label` value is what you will see in the tasks list. Name this whatever you like.

The `detail` value is what you will see as the description of the task in the tasks list. It's highly recommended to rename this value to differentiate it from similar tasks.

The `problemMatcher` value selects the output parser to use for finding errors and warnings in the compiler output. For `clang++`, you'll get the best results if you use the `$gcc` problem matcher.

From now on, the play button will read from `tasks.json` to figure out how to build and run your program. You can define multiple build tasks in `tasks.json`, and whichever task is marked as the default will be used by the play button. In case you need to change the default compiler, you can run **Tasks: Configure default build task**. Alternatively you can modify the `tasks.json` file and remove the default by replacing this segment:

```
"group": {  
  "kind": "build",  
  "isDefault": true  
},
```

with this:

```
"group": "build",
```

## Modifying tasks.json

You can modify your `tasks.json` to build multiple C++ files by using an argument like `"${workspaceFolder}/*.cpp"` instead of `${file}`. This will build all `.cpp` files in your current folder. You can also modify the output filename by replacing `"${fileDirname}/${fileBasenameNoExtension}"` with a hard-coded filename (for example `"${workspaceFolder}/myProgram.out"`).

## Debug helloworld.cpp

1. Go back to `helloworld.cpp` so that it is the active file.
2. Set a breakpoint by clicking on the editor margin or using `F9` on the current line.



```

helloworld.cpp x
helloworld.cpp > main()
1  #include <iostream>
2  #include <vector>
3  #include <string>
4
5  using namespace std;
6
7  int main()
8  {
9      vector<string> msg{"Hello", "C++", "World", "from", "VS Code", "and the C++ extension!"};
10     for (const string &word : msg)
11     {
12         cout << word << " ";
13     }
14     cout << endl;
15 }

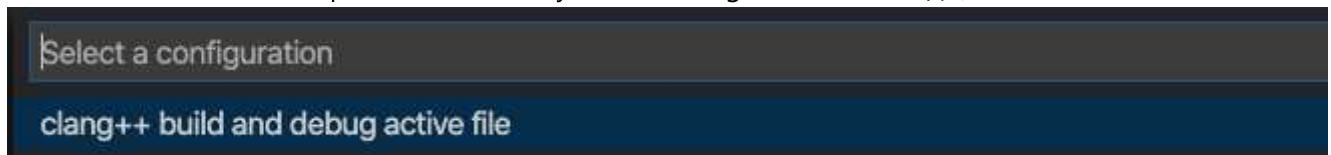
```

3. From the drop-down next to the play button, select **Debug C/C++ File**.

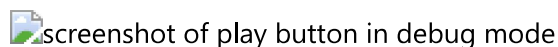
 Screenshot of play button drop-down



4. Choose **clang++ build and debug active file** from the list of detected compilers on your system (you'll only be asked to choose a compiler the first time you run/debug helloworld.cpp).



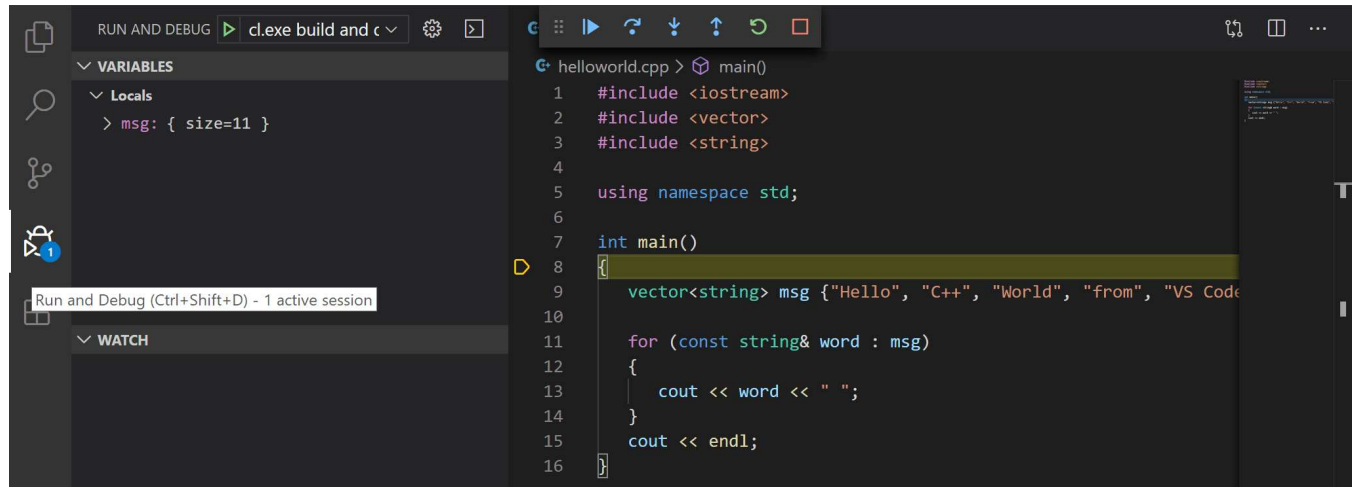
The play button has two modes: **Run C/C++ File** and **Debug C/C++ File**. It will default to the last-used mode. If you see the debug icon in the play button, you can just click the play button to debug, instead of selecting the drop-down menu item.

 Screenshot of play button in debug mode

## Explore the debugger

Before you start stepping through the code, let's take a moment to notice several changes in the user interface:

- The Integrated Terminal appears at the bottom of the source code editor. In the **Debug Output** tab, you see output that indicates the debugger is up and running.
- The editor highlights the first statement in the `main` method. This is a breakpoint that the C++ extension automatically sets for you:



- The **Run and Debug** view on the left shows debugging information. You'll see an example later in the tutorial.
- At the top of the code editor, a debugging control panel appears. You can move this around the screen by grabbing the dots on the left side.

## Step through the code

Now you're ready to start stepping through the code.

1. Click or press the **Step over** icon in the debugging control panel so that the `for (const string& word : msg)` statement is highlighted.



The **Step Over** command skips over all the internal function calls within the `vector` and `string` classes that are invoked when the `msg` variable is created and initialized. Notice the change in the **Variables** window. The contents of `msg` are visible because that statement has completed.

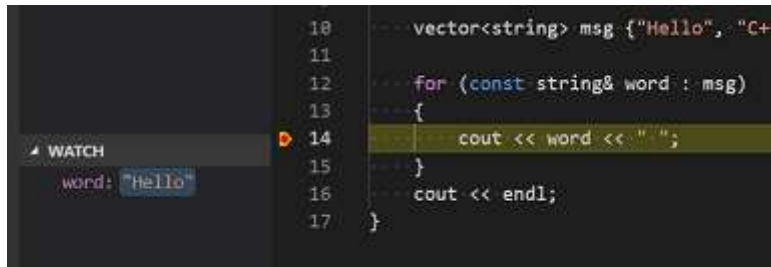
2. Press **Step over** again to advance to the next statement (skipping over all the internal code that is executed to initialize the loop). Now, the **Variables** window shows information about the loop variable.
3. Press **Step over** again to execute the `cout` statement. **Note** As of the March 2019 version of the extension, no output will appear in the **DEBUG CONSOLE** until the last `cout` completes.



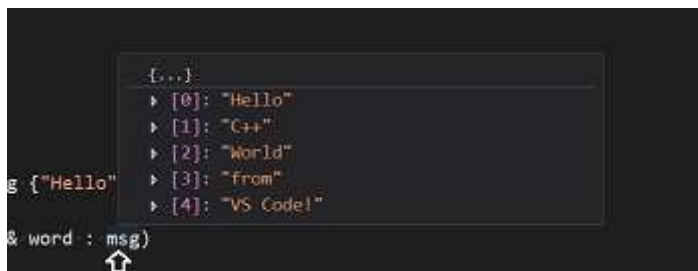
## Set a watch

You might want to keep track of the value of a variable as your program executes. You can do this by setting a **watch** on the variable.

1. Place the insertion point inside the loop. In the **Watch** window, click the plus sign and in the text box, type `word`, which is the name of the loop variable. Now view the **Watch** window as you step through the loop.



2. To quickly view the value of any variable while execution is paused, you can hover over it with the mouse pointer.

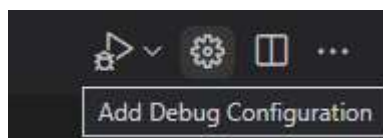


## Customize debugging with launch.json

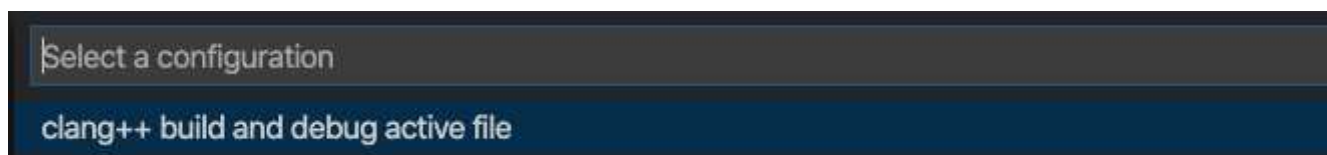
When you debug with the play button or `F5`, the C++ extension creates a dynamic debug configuration on the fly.

There are cases where you'd want to customize your debug configuration, such as specifying arguments to pass to the program at runtime. You can define custom debug configurations in a `launch.json` file.

To create `launch.json`, choose **Add Debug Configuration** from the play button drop-down menu.



You'll then see a dropdown for various predefined debugging configurations. Choose **clang++ build and debug active file**.



VS Code creates a `launch.json` file, which looks something like this:

```

{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "C/C++: clang++ build and debug active file",
      "type": "cppdbg",
      "request": "launch",
      "program": "${fileDirname}/${fileBasenameNoExtension}",
      "args": [],
      "stopAtEntry": true,
      "cwd": "${workspaceFolder}",
      "environment": [],
      "externalConsole": false,
      "MIMode": "lldb",
      "preLaunchTask": "C/C++: clang++ build active file"
    }
  ]
}

```

The `program` setting specifies the program you want to debug. Here it is set to the active file folder `${fileDirname}` and active filename `${fileBasenameNoExtension}`, which if `helloworld.cpp` is the active file will be `helloworld`. The `args` property is an array of arguments to pass to the program at runtime.

By default, the C++ extension won't add any breakpoints to your source code and the `stopAtEntry` value is set to `false`.

Change the `stopAtEntry` value to `true` to cause the debugger to stop on the `main` method when you start debugging.

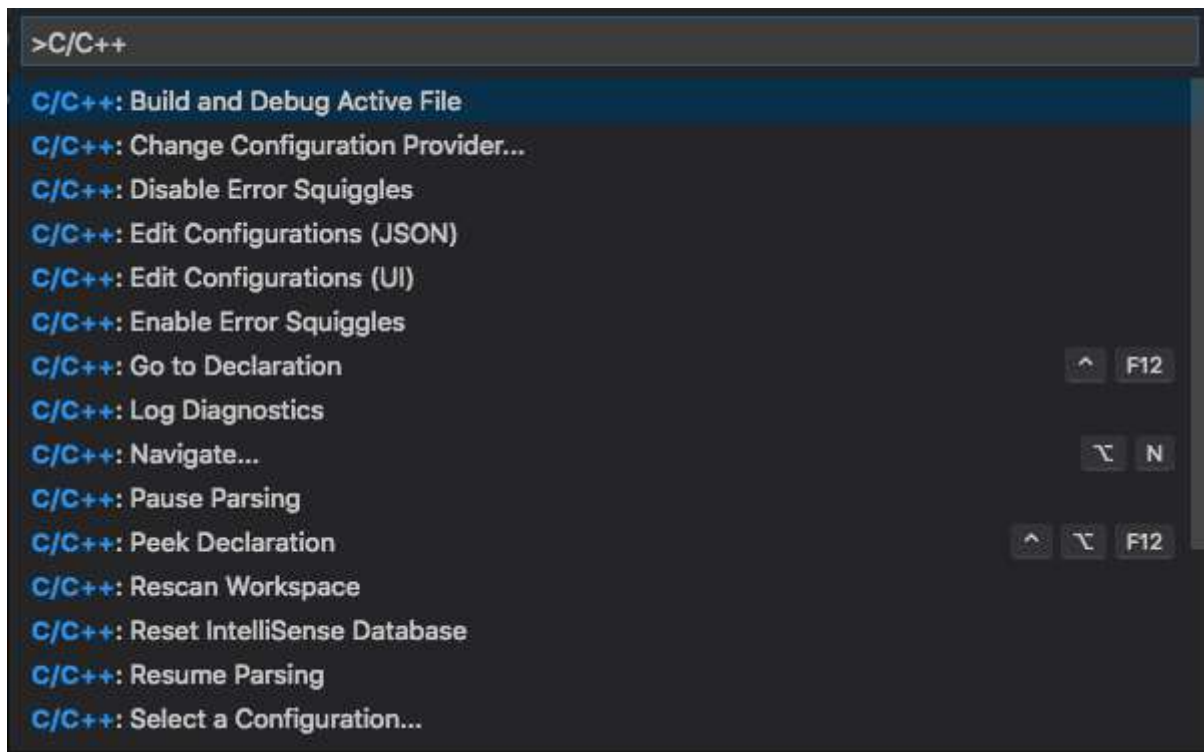
Ensure that the `preLaunchTask` value matches the `label` of the build task in the `tasks.json` file.

From now on, the play button and `F5` will read from your `launch.json` file when launching your program for debugging.

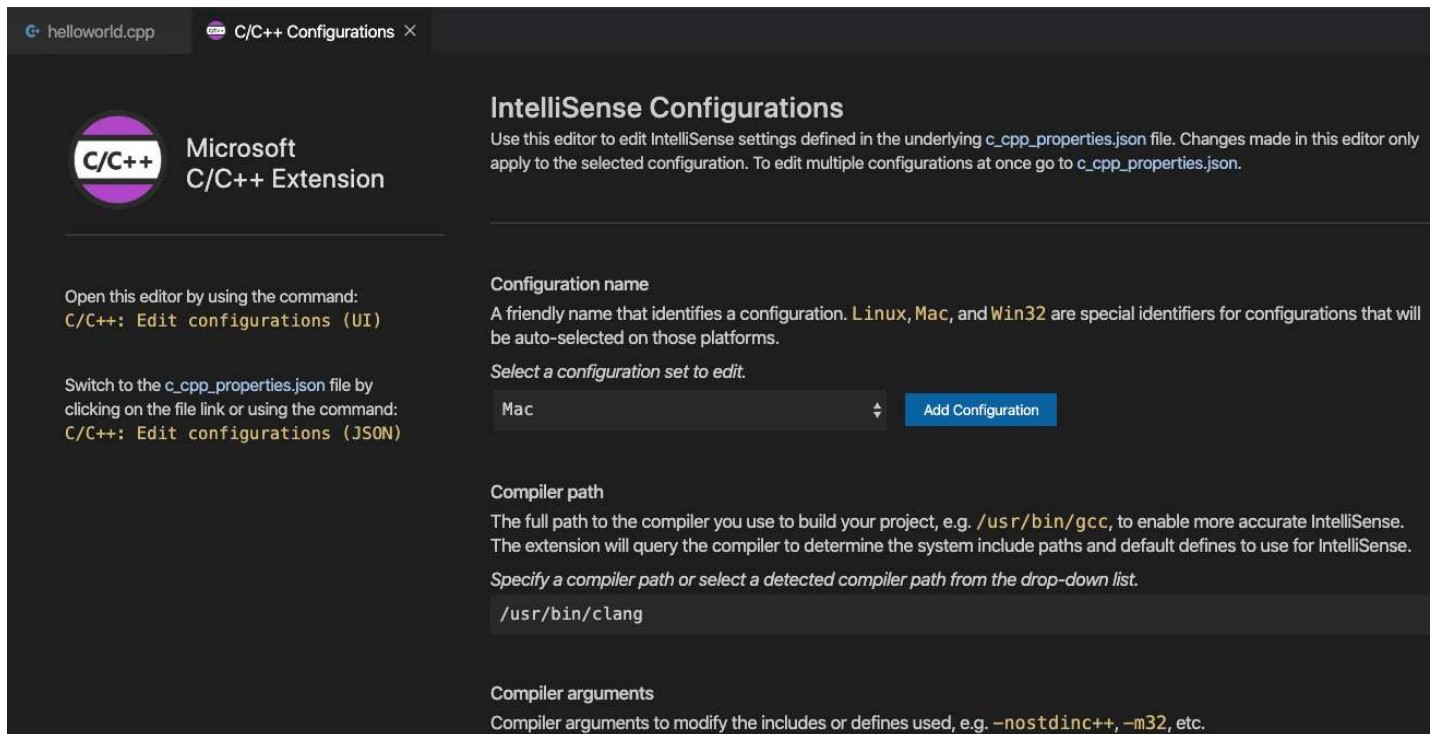
## C/C++ configuration

For more control over the C/C++ extension, create a `c_cpp_properties.json` file, which allows you to change settings such as the path to the compiler, include paths, which C++ standard to compile against (such as C++17), and more.

View the C/C++ configuration UI by running the command **C/C++: Edit Configurations (UI)** from the Command Palette (`Ctrl+Shift+P`).



This opens the **C/C++ Configurations** page.



Visual Studio Code places these settings in `.vscode/c_cpp_properties.json`. If you open that file directly, it should look something like this:

```
{
  "configurations": [
    {
      "name": "Mac",
      "includePath": ["${workspaceFolder}/**"],
      "defines": [],
      "macFrameworkPath": [
        "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/System/Library/Frameworks"
      ],
      "compilerPath": "/usr/bin/clang",
      "cStandard": "c11",
      "cppStandard": "c++17",
      "intelliSenseMode": "clang-x64"
    }
  ],
  "version": 4
}
```

You only need to modify the **Include path** setting if your program includes header files that are not in your workspace or the standard library path.

### Compiler path

`compilerPath` is an important configuration setting. The extension uses it to infer the path to the C++ standard library header files. When the extension knows where to find those files, it can provide useful features like smart completions and **Go to Definition** navigation.

The C/C++ extension attempts to populate `compilerPath` with the default compiler location based on what it finds on your system. The `compilerPath` search order is:

- Your `PATH` for the names of known compilers. The order the compilers appear in the list depends on your `PATH`.
- Then hard-coded Xcode paths are searched, such as  
`/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/`

### Mac framework path

On the C/C++ Configuration screen, scroll down and expand **Advanced Settings** and ensure that **Mac framework path** points to the system header files. For example:

```
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/System/Library/Frameworks
```

### Reusing your C++ configuration

VS Code is now configured to use Clang on macOS. The configuration applies to the current workspace. To reuse the configuration, just copy the JSON files to a `.vscode` folder in a new project folder (workspace) and change the names of the source file(s) and executable as needed.

## Troubleshooting

### Compiler and linking errors

The most common cause of errors (such as `undefined _main`, or attempting to link with file built for unknown-unsupported file format, and so on) occurs when `helloworld.cpp` is not the active file when you start a build or start debugging. This is because the compiler is trying to compile something that isn't source code, like your `launch.json`, `tasks.json`, or `c_cpp_properties.json` file.

If you see build errors mentioning "C++11 extensions", you may not have updated your `tasks.json` build task to use the `clang++` argument `--std=c++17`. By default, `clang++` uses the C++98 standard, which doesn't support the initialization used in `helloworld.cpp`. Make sure to replace the entire contents of your `tasks.json` file with the code block provided in the Run `helloworld.cpp` section.

### Terminal won't launch For input

On macOS Catalina and onwards, you might have a issue where you are unable to enter input, even after setting `"externalConsole": true`. A terminal window opens, but it does not actually allow you type any input.

The issue is currently tracked #5079 (<https://github.com/microsoft/vscode-cpptools/issues/5079>).

The workaround is to have VS Code launch the terminal once. You can do this by adding and running this task in your `tasks.json`:

```
{
  "label": "Open Terminal",
  "type": "shell",
  "command": "osascript -e 'tell application \"Terminal\"\\ndo script \"echo hello\"\\nend tell'",
  "problemMatcher": []
}
```

You can run this specific task using **Terminal > Run Task...** and select **Open Terminal**.

Once you accept the permission request, then the external console should appear when you debug.

### Next steps

- Explore the VS Code User Guide (</docs/editor/codebasics>).
- Review the Overview of the C++ extension (</docs/languages/cpp>)
- Create a new workspace, copy your `.json` files to it, adjust the necessary settings for the new workspace path, program name, and so on, and start coding!

## Was this documentation helpful?

Yes

No


---

5/13/2022

 [Subscribe\(/feed.xml\)](/feed.xml)  [Ask questions\(https://stackoverflow.com/questions/tagged/vscode\)](https://stackoverflow.com/questions/tagged/vscode)

 [Follow @code\(https://go.microsoft.com/fwlink/?LinkID=533687\)](https://go.microsoft.com/fwlink/?LinkID=533687)

 [Request features\(https://go.microsoft.com/fwlink/?LinkID=533482\)](https://go.microsoft.com/fwlink/?LinkID=533482)

 [Report issues\(https://www.github.com/Microsoft/vscode/issues\)](https://www.github.com/Microsoft/vscode/issues)


 [Watch videos\(https://www.youtube.com/channel/UCs5Y5\\_7XK8HLDX0SLNwkd3w\)](https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

Hello from Seattle. [Follow @code \(https://go.microsoft.com/fwlink/?LinkID=533687\)](https://go.microsoft.com/fwlink/?LinkID=533687)

[Support \(https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d\)](https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d)

[Privacy \(https://go.microsoft.com/fwlink/?LinkId=521839\)](https://go.microsoft.com/fwlink/?LinkId=521839) [Terms of Use \(https://www.microsoft.com/legal/terms-of-use\)](https://www.microsoft.com/legal/terms-of-use)

[License \(/License\)](/License)

 [Microsoft \(https://www.microsoft.com\)](https://www.microsoft.com)

© 2023 Microsoft