

Visual Studio Community - New Users

This document provides a starting point for new users to get familiar with the Visual Studio Community IDE.

Overview

Visual Studio Community is a free, fully-featured, and extensible IDE for creating modern applications for Windows, Android, and iOS, as well as web applications and cloud services. This article will walk through how to start building any kind of application inside of Visual Studio IDE.

Contents

1. Requirements
 2. Installing Visual Studio Community
 3. Open and login to Visual Studio
 4. Create, build, and run your first project
 5. Edit your code
 6. Debug your application
 7. Testing your application
 8. Visual Studio Extensions
 9. Share code with Visual Studio Team Services and Github
-

Requirements

For an up-to-date list of requirements, go to the [Visual Studio 2015 Requirements](#) to view the system requirements for Visual Studio 2015 Community Edition.

Installing Visual Studio Community

The Visual Studio Community installer is found [here](#). This will install all of the Visual Studio bits and dependencies.

You will have the opportunity to install the default options, or to customize your install when you first run the Visual Studio installer:

Visual Studio installer default versus customize

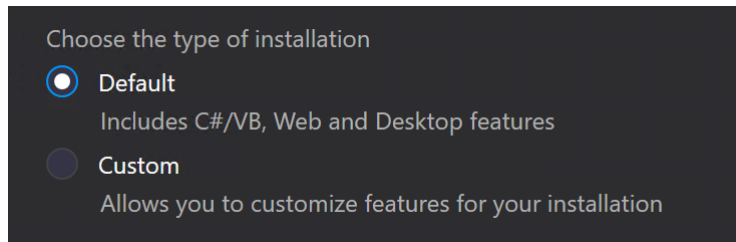


Figure 1: Visual Studio installer choice

If you choose **Default**, then Visual Studio 2015 Community will be installed with the default settings.

Otherwise, you will be able to choose which features to install if you choose the **Custom** route.

Once complete, Visual Studio 2015 Community IDE should be installed on your system.

Note : Full Visual Studio 2015 installation details found [here](#).

Open and login to Visual Studio

Visual Studio is a feature-rich IDE with many integrations into services that further enhance the development experience. By logging into Visual Studio with your Microsoft Account, you connect your IDE to those services and gain access to Visual Studio Dev Essentials. This will let you start using Azure credits, publish code to private repositories, sync your settings, and gain access to all the learning material available through Dev Essentials.

Once Visual Studio IDE has been installed, you will want to open it up and login. This connects you to additional services that require your Microsoft account.

Login with your Microsoft Account

Once logged in, Visual Studio will finish up a few configuration steps to make the IDE ready for your use.

Note: You can always log in at a later time and skip this step.

You will then be brought to the start page of the Visual Studio IDE:

Visual Studio's start page

The start page has up-to-date content that will keep you updated on news, link out to external learning resources, and quick links to recent projects and files.

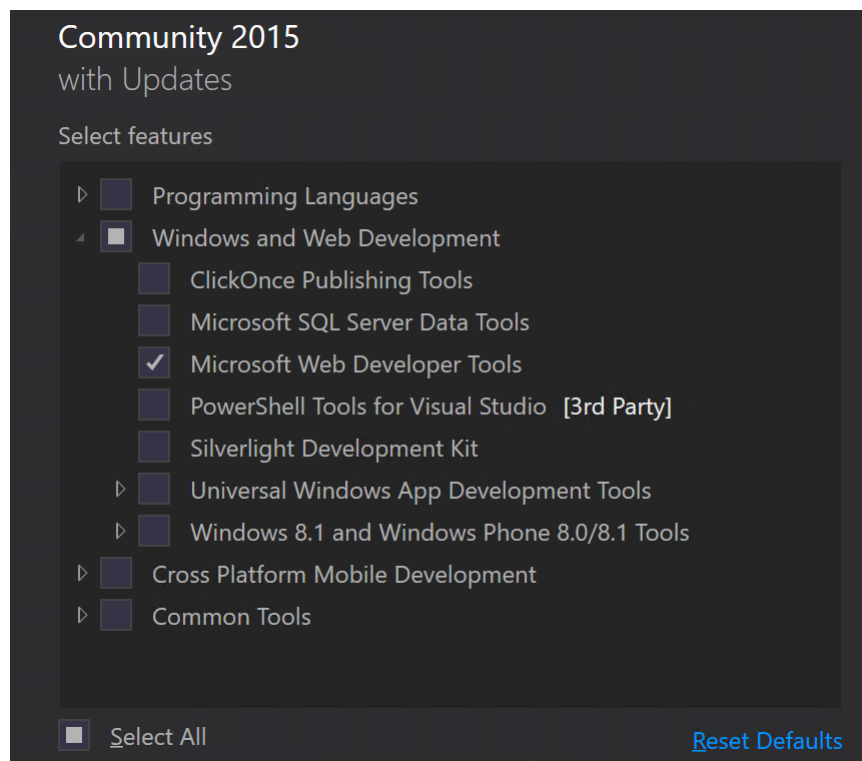


Figure 2: Visual Studio installer feature selection

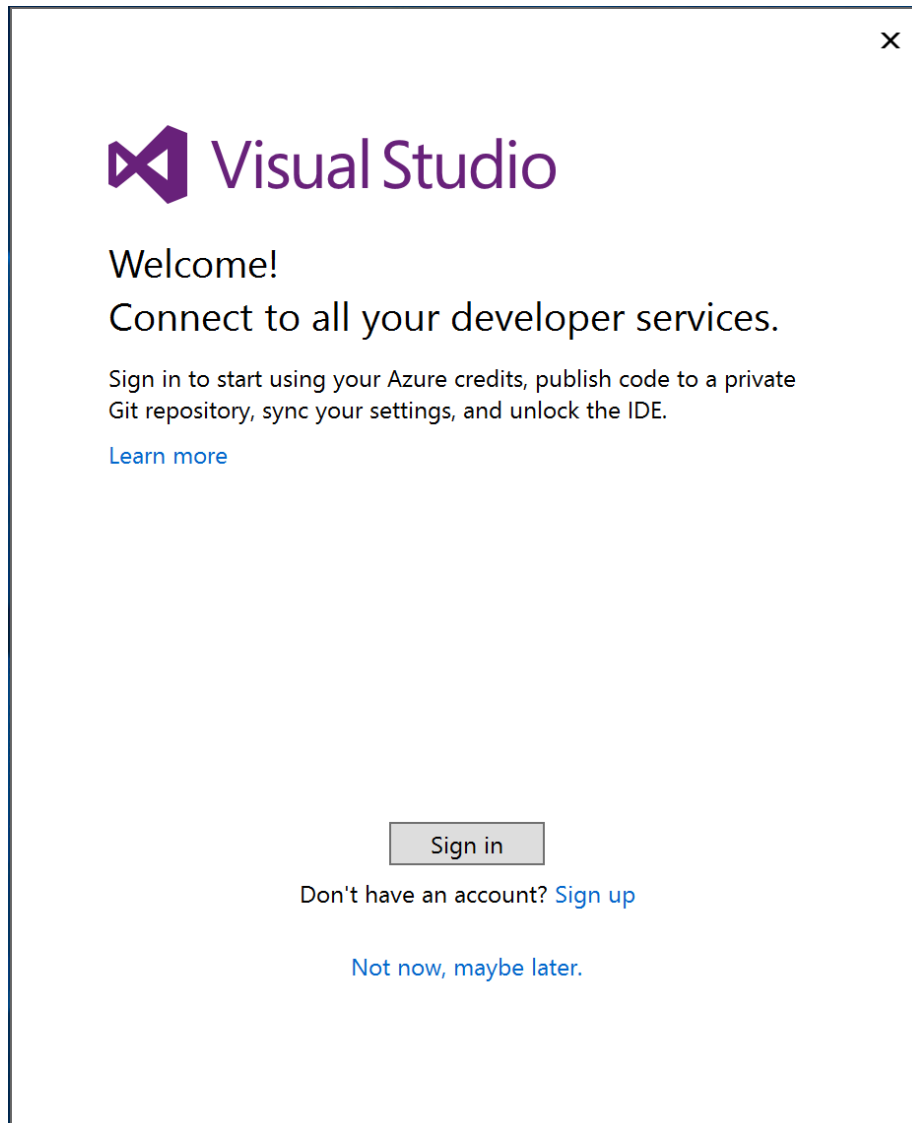


Figure 3: Sign in splash screen

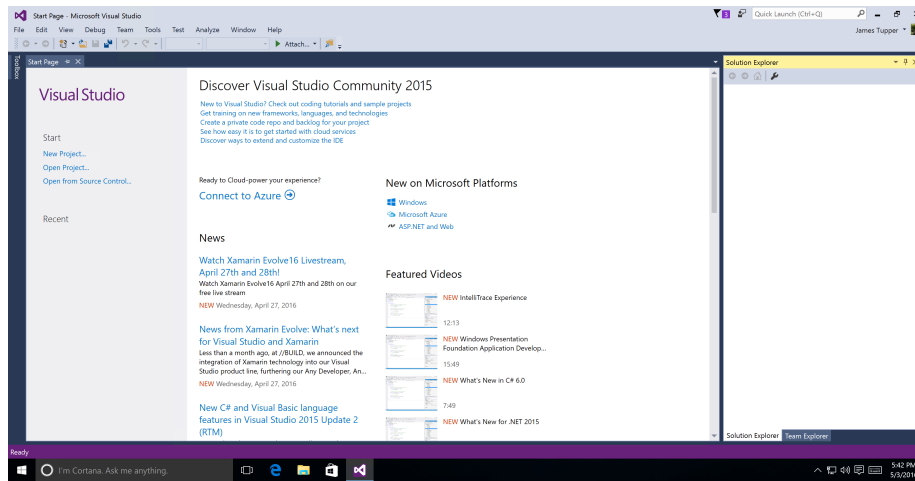


Figure 4: Visual Studio IDE start page

Note: For more information on the various components of Visual Studio IDE, go to the [Visual Studio IDE documentation page](#).

Create, build, and run your first project

When you create an app, application, website, Web App, script, plug-in, etc in Visual Studio, you start with a project. In a logical sense, a project contains of all the source code files, icons, images, data files and anything else that will be compiled into an executable program or web site, or else is needed in order to perform the compilation. A project also contains all the compiler settings and other configuration files that might be needed by various services or components that your program will communicate with.

A project is contained, in a logical sense and in the file system, within a solution, which may contain one or more projects, along with build information, Visual Studio window settings, and any miscellaneous files that aren't associated with any project. In a literal sense, the solution is a text file with its own unique format; it is generally not intended to be edited by hand.

Visual Studio projects can range from their different types (web, desktop, service, etc.) to different languages (C#, F#, Python, TypeScript, etc.), but they all use the same Visual Studio IDE features to develop, run, test, and debug each respective project. You can browse all the various types of projects within the New Project wizard, and any project templates that aren't installed will prompt you to download them first.

Note: For further reading on projects and solutions, go to the Projects and Solutions in Visual Studio article.

Clicking on **File / New / Project** will bring up the New Project Wizard.

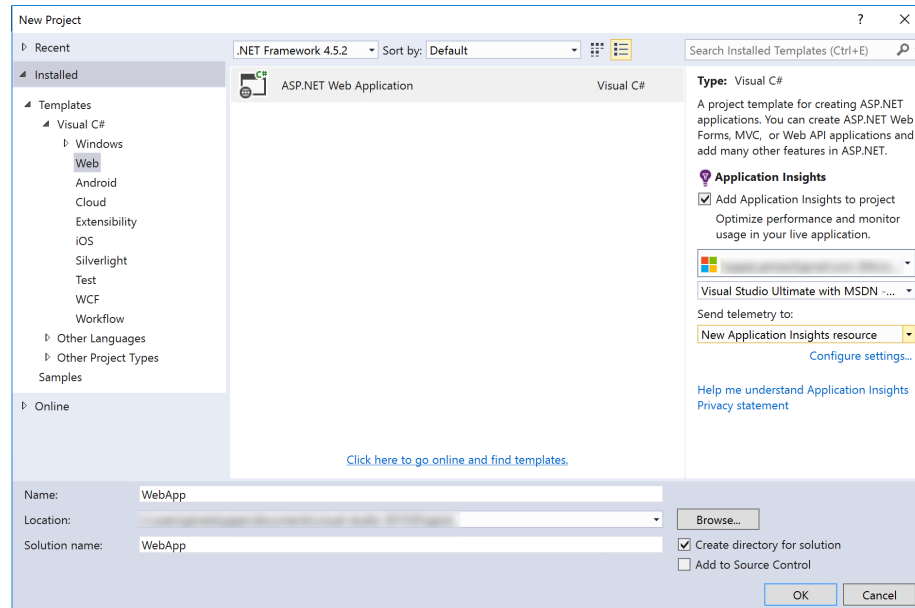


Figure 5: Create new project

*Choose a name and then click **OK***

This will build out the folder structure and files, and a starting point for the particular project type. Once finished, it should take you to the **Solution Explorer**, and open the starting code page.

Note: Some project types need more configuration via wizard pages to finish.

Initial files and structure for new project

Note: This project template is an ASP.NET Web Application. For more information on creating Web Applications, go to the ASP.NET Documentation site.

Note: It is simple to host Web Applications in Azure with the many integrations from Visual Studio IDE. For further reading, go to Azure Documentation Center.

Notice that all the code files, and dependencies, in the Solution Explorer on the right hand side of the Window.

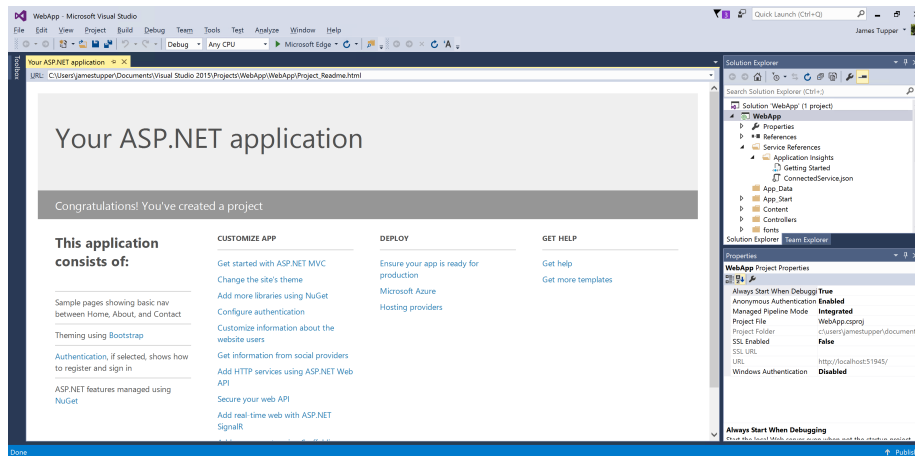


Figure 6: Initial files for new project

The first thing any developer will want to do is build their application to make sure it successfully builds. To do so, click **Build / Build Solution** or **F6**. Then click **View / Output** to open the output window.

Initial solution build

In the output window, ensure the output is from **Build**. Notice that the output of the compilation of your project is logged in this window. The blue status bar at the bottom also shows that the build has succeeded. You will see this bar change color depending on the status of the build (i.e. running, succeeded, failed).

At the top of the screen, there is a green play button - this will launch your application for you and attach the debugger to it.

Run your application

Click the debugger button to launch your application.

Note: This is a web application project, so you have the choice to choose from various web browsers - i.e. Microsoft Edge.

Running web application

You will see that Visual Studio IDE switches to a debugging view while your application is running. Interact with your application as you see fit, and find the **red, square button** to stop the running application.

Stop your application

This will now bring you back to the screen prior to running your application. You have now created a new project in Visual Studio IDE and was able to run and stop it.

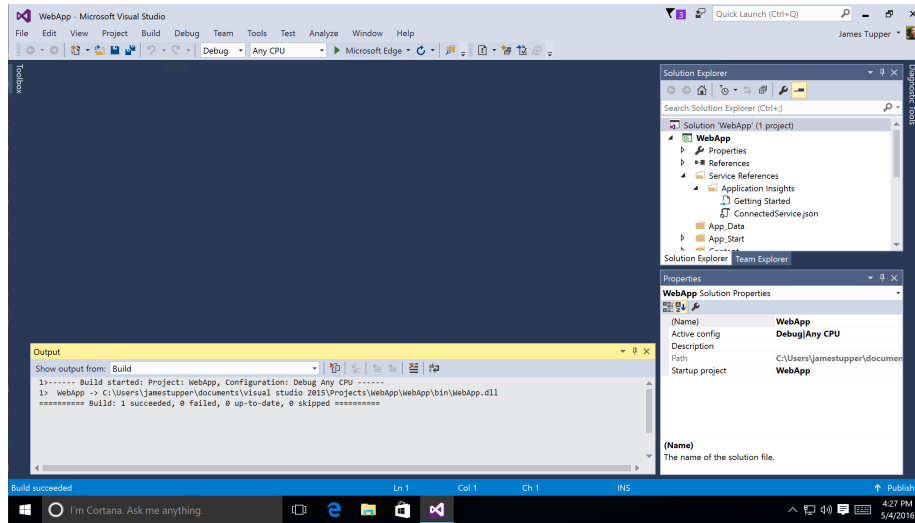


Figure 7: Initial solution build

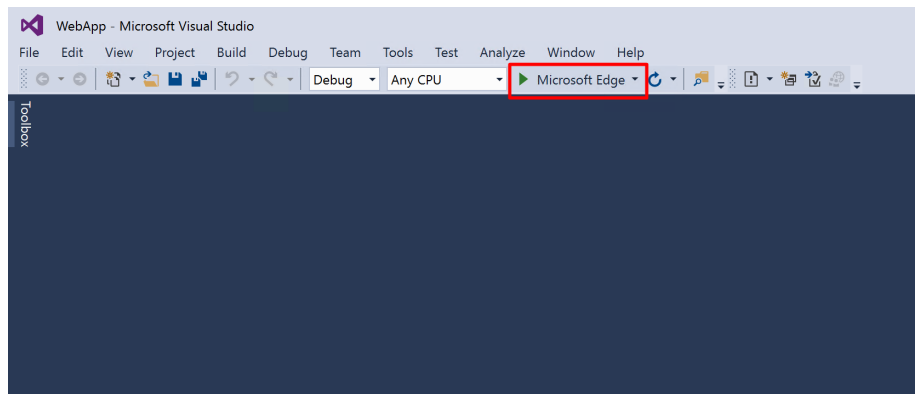


Figure 8: Run your application

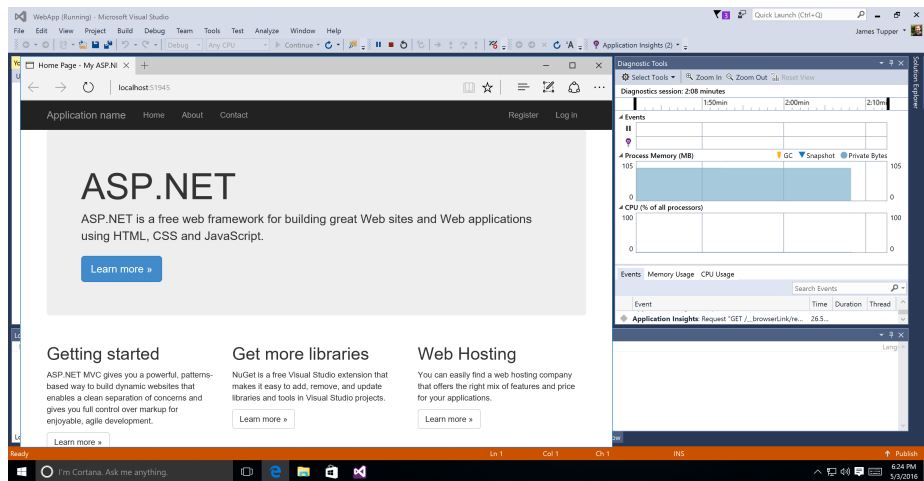


Figure 9: Running web application

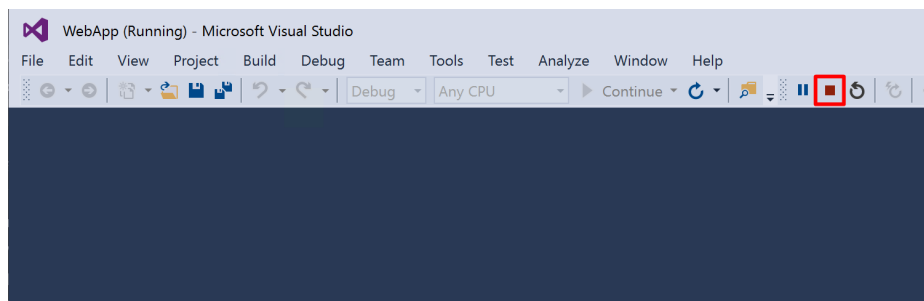


Figure 10: Stop your application

For further reading on getting started with various types of applications, go to the [Get Started Developing with Visual Studio](#) article.

Edit your code

For any developer, editing code is a core piece of functionality that is required from any IDE. Visual Studio excels in aiding developers in writing code, navigating, and fixing issues. From giving you confidence with tools such as Intellisense and auto completion to navigating through your code without losing context, Visual Studio has a wide range of features and extensions to add effectiveness and efficiency to developers.

Within your application, **open a code file and start typing anywhere**. You should notice context menus appearing as you type to describe APIs and to help with auto-completion.

[Intellisense]](images/edit__intellisense.png “Intellisense”)

Intellisense in Visual Studio

You will also notice that there is syntax highlighting which works with 20 languages out of box with Visual Studio (more languages supported extensions) and squiggly lines to point out syntactical problems with your code.

If you want to dive into code you’re writing a bit more without losing context, you can use the *Peek to definition* feature by **Right-clicking / Peek Definition** on the method you want, or by pressing **Alt+F12**.

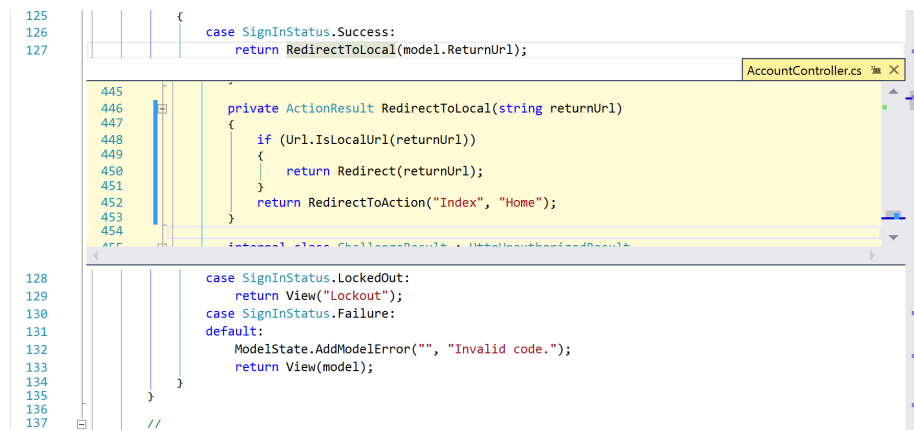


Figure 11: Peek definition in Visual Studio

Peek definition in Visual Studio

These are just a couple of features that help developers effectively edit their code. Code Lens helps understand changing code, light bulb icons help identify common coding issues, built-in refactoring options for multiple languages, and so on. This lets Visual Studio help its users create the best code they can.

Note: For further reading on editing code in Visual Studio, visit the [Writing Code in the Code and Text Editor](#).

Debug your application

The Visual Studio debugger helps you observe the run-time behavior of your program and find problems. The debugger works with all Visual Studio programming languages and their associated libraries. With the debugger, you can break execution of your program to examine your code, examine and edit variables, view registers, see the instructions created from your source code, and view the memory space used by your application.

Paste the following code into an empty Console Application:

```
static void Main(string[] args){
    int testInt = 0;
    for (int i = 0; i <= 10; i++){
        testInt += 1;
    }
    Console.WriteLine(testInt);
}
```

Click on the green play button that says Start, to launch your application. You will note that the window opens and closes before you are able to see the results in the console window.

Let's add a breakpoint on the line where we write to the Console:

Add a breakpoint where we output to the console

To add a breakpoint, put your cursor on the same line as the `Console.WriteLine` and click **Debug / New Breakpoint / Function Breakpoint**.

Note: You can also click in the left margin at the same line, or press F9 to add a breakpoint.

Start debugging again by clicking the green start button, and you will notice that execution pauses prior to the `Console.WriteLine` code executes.

Stop at breakpoint

Execute the selected line of code (click **Debug / Step Over** or **F10**). The curly brace on the next line should now be highlighted in yellow, and the console window should have the value **10** printed in it.

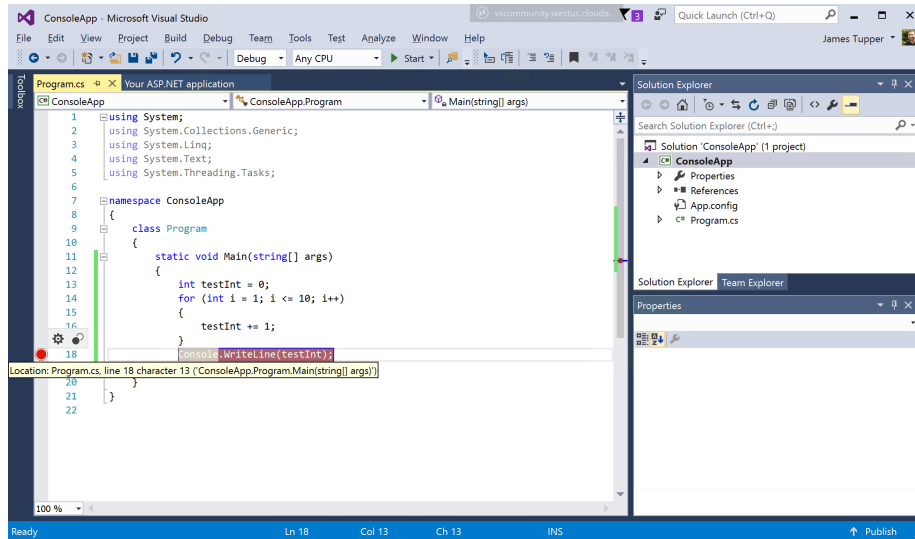


Figure 12: Add a breakpoint where we output to the console

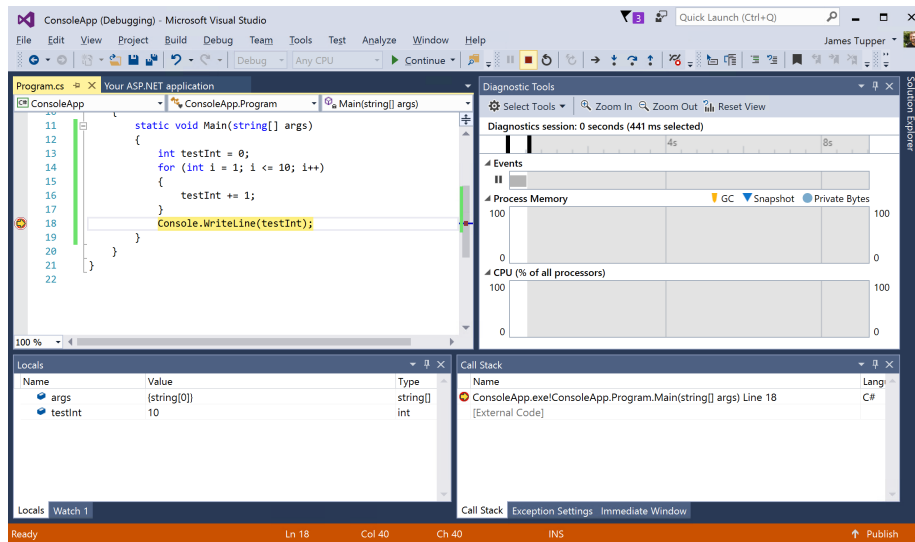


Figure 13: Stop at breakpoint

Now let's look at the variable values. Just below the code you should see the **Locals** window which shows the current value of local variables. The window shows **testInt** with a value of **10**.

Stop debugging (click **Debug / Stop Debugging** or **SHIFT + F5**).

For further reading on debugging, go to the [Debugging in Visual Studio](#) article.

Testing your application

Check that your code is working as expected by creating and running unit tests. It's called unit testing because you break down the functionality of your program into discrete testable behaviors that you can test as individual units. Visual Studio Test Explorer provides a flexible and efficient way to run your unit tests and view their results in Visual Studio. Visual Studio installs the Microsoft unit testing frameworks for managed and native code. Use a unit testing framework to create unit tests, run them, and report the results of these tests. Rerun unit tests when you make changes to test that your code is still working correctly. When you use Visual Studio Enterprise, you can run tests automatically after every build.

Unit testing has the greatest effect on the quality of your code when it's an integral part of your software development workflow. As soon as you write a function or other block of application code, create unit tests that verify the behavior of the code in response to standard, boundary, and incorrect cases of input data, and that check any explicit or implicit assumptions made by the code. With test driven development, you create the unit tests before you write the code, so you use the unit tests as both design documentation and functional specifications.

Note: For more information on unit testing, visit the [Unit Test Your Code](#) article.

The first thing we need to do is add a test project to our solution. To do so, go to **File / New / Project**. On the left side choose **Test** and select the **Unit Test Project**. Click **OK** to create the project.

Create unit test project

Once created, you will be presented with a new unit testing file. Add the following code into the test method:

```
// arrange
var x = 5;
var y = 7;
```

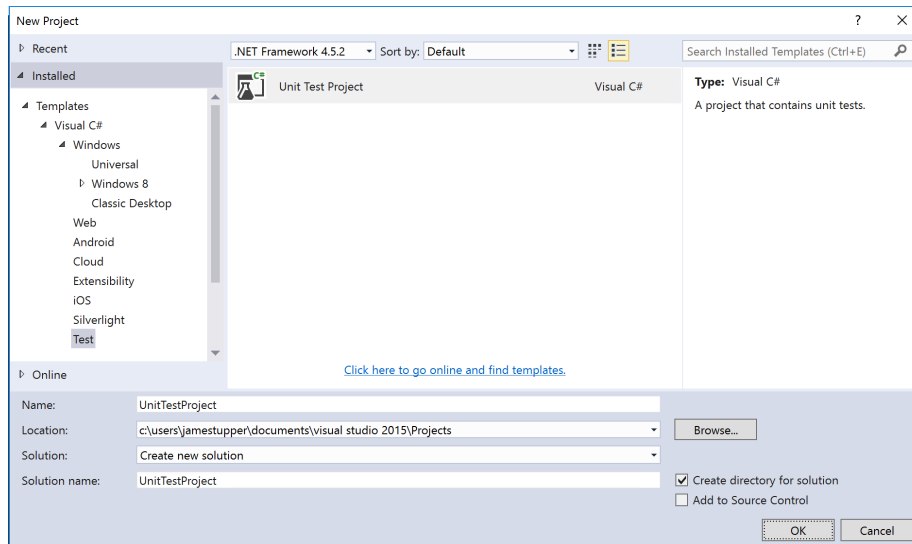


Figure 14: Create unit test project

```
// act
var sum = x + y;

// assert
Assert.AreEqual(12, sum);
```

You should end up with something that looks like this:

Add code snippet to unit test method

Make sure that there are no build errors by building your solution (**Build / Build Solution** or **F6**). After your solution builds successfully, click **Test / Run / All Tests** to build the solution and then run the unit tests.

Run unit tests

You will notice that the *Test Explorer* window appears when you do this. This is where the result for automated tests will be shown. Adding test classes and methods are just as easy as writing code, and they are automatically discovered on every build and populated in the Test Explorer.

Note: for more information on the test explorer, read the Run unit tests with Test Explorer article.

The previous example uses the built-in framework, *MSTest*. It is just as easy to use third-party frameworks (i.e. NUnit and xUnit) via extensions. Read Install third-party unit test frameworks for more information.

Note: For further information on unit testing, read the Unit Test

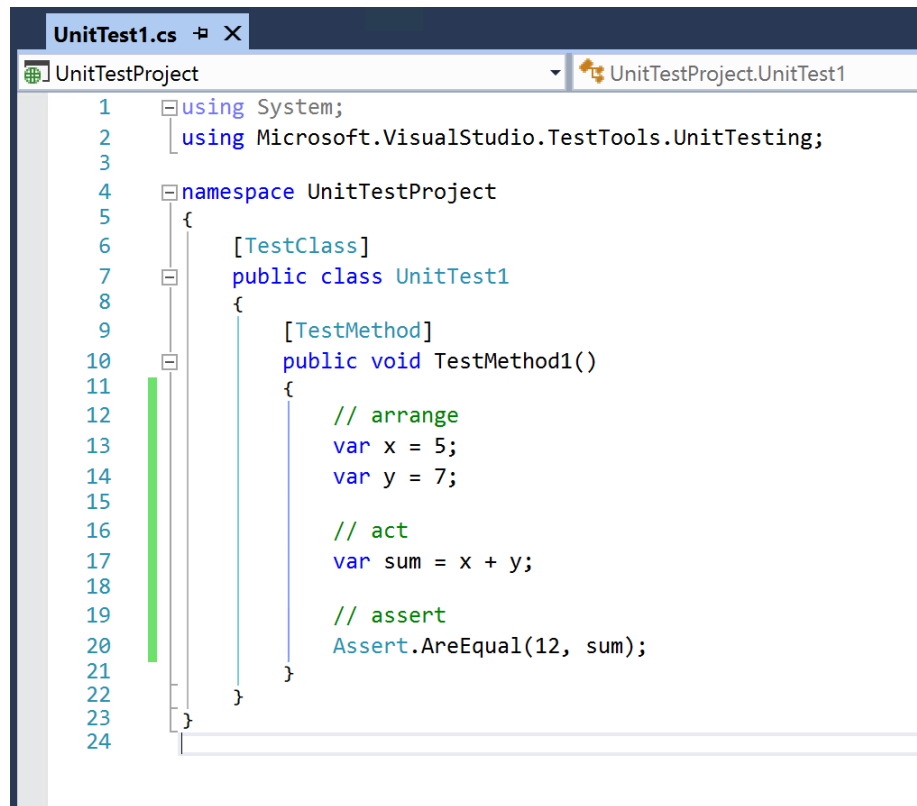


Figure 15: Add code snippet to unit test method

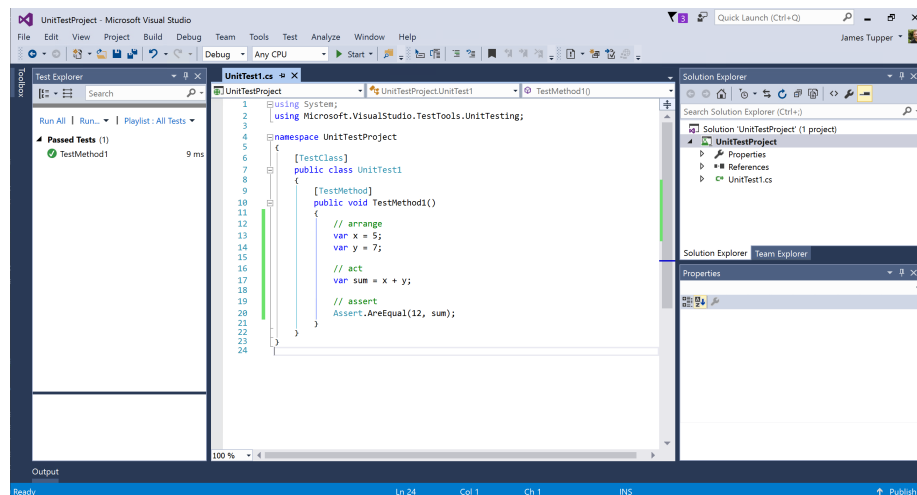


Figure 16: Run unit tests

Your Code article.

Visual Studio Extensions

Visual Studio Extensions allow developers to create custom tools, controls, and templates to help get the most out of Visual Studio IDE. There are currently thousands of extensions that can help improve your productivity. These extensions allow you to use different languages, build applications for mobile devices, enhance your productivity in the tool, and so on. The possibilities are limitless.

Note: To browse Visual Studio Extensions, go to the Visual Studio Gallery.

To open the Extensions and Updates window, click **Tools / Extensions and Updates...** This window gives you the ability to view all the installed extensions, find extensions online, and update any out-of-date extensions.

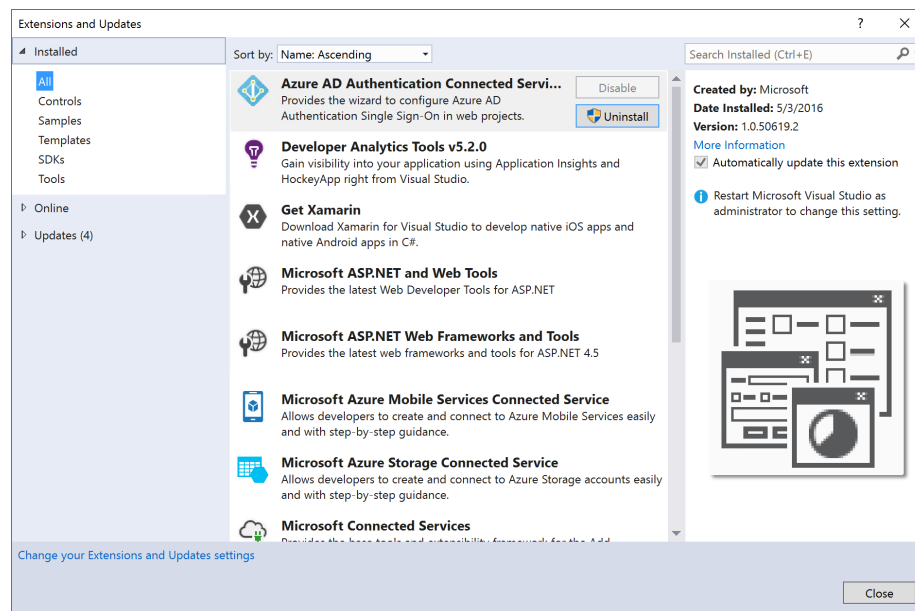


Figure 17: Updates and Extensions - Installed

Updates and Extensions - Installed

Let's go ahead and install the Productivity Power Tools 2015. Click **Online** and then search for **productivity** in the search menu. The first option should be *Productivity Power Tools 2015*, click the **Download** button to install.

Download Productivity Tools Extension

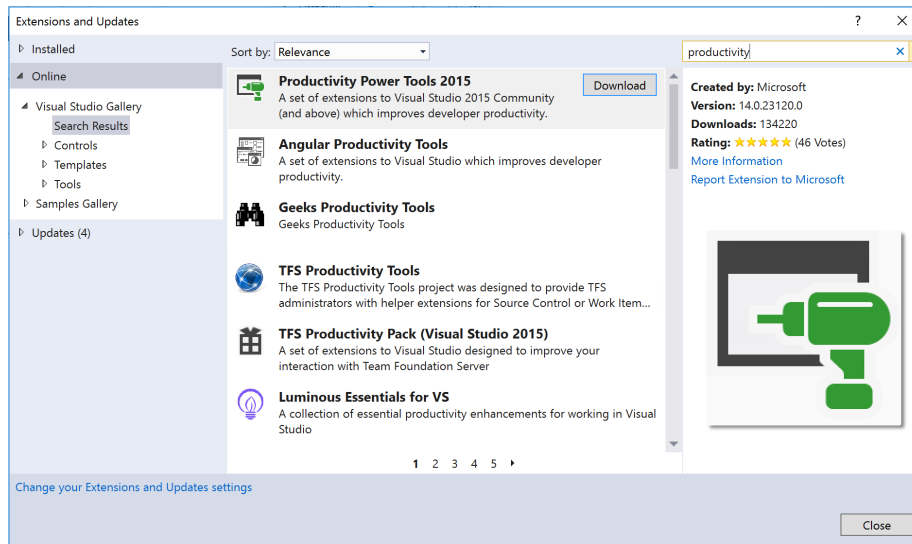


Figure 18: Download Productivity Tools Extension

Once installed, Visual Studio will notify you that you need to restart the IDE in order for the extension to load. **Restart Visual Studio** for the Productivity Power Tools extension to work.

As you can see, extensions are an easy way to add functionality all over Visual Studio. For further reading on how to extend Visual Studio yourself, check out the Extensibility in Visual Studio article.

To browse for current extensions, go to the Visual Studio Gallery

To browse open source extensions, go to Community Extensions page on Github.

You can also check out the Visual Studio Marketplace which is coming soon.

Share code with Visual Studio Team Services (VSTS)

Very often, developers have the needs to share the code with other developers, back up their code, version their code, build their code, test their code, and so on. Visual Studio IDE has integrations into all of these services in order to enhance and promote team productivity and quality software.

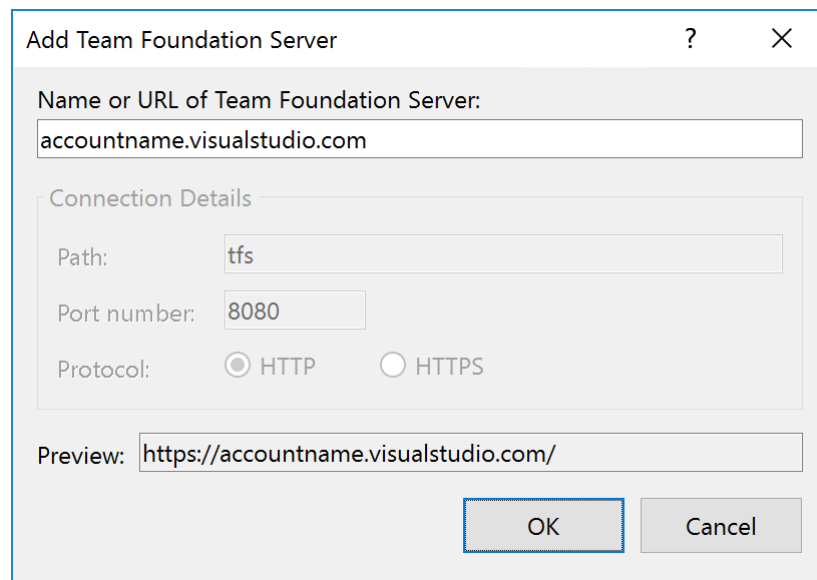
There are a multitude of various ways to this, but we are going to stick with Git as the version control and Visual Studio Team Services (VSTS) as the online

provider to host the code.

Note: Here is more reading for VSTS, Github, Git, Azure, and Devops and Application Lifecycle Management.

Open **Visual Studio Community** and navigate to the *Team Explorer* page which can be opened by **View / Team Explorer**. By default, you will be able to connect and interact with VSTS.

If you don't already have a VSTS account, click the **Get started for free** link to set one up. Then, click the **Connect...** / **Servers** / **Add** buttons to add a VSTS account.



Add Team Foundation Server

Name or URL of Team Foundation Server:
accountname.visualstudio.com

Connection Details

Path: tfs

Port number: 8080

Protocol: ☒ HTTP ☐ HTTPS

Preview: https://accountname.visualstudio.com/

OK Cancel

Figure 19: Add a VSTS account

Add a VSTS account

Once added, you can now choose the *Team Project* that you want and click **Connect** to connect to it. Once connected, you will be able to share your code and interact with all of the other services available in VSTS.

Connected to a Team Project in VSTS

Note: Here is a link to the Github Extension for Visual Studio

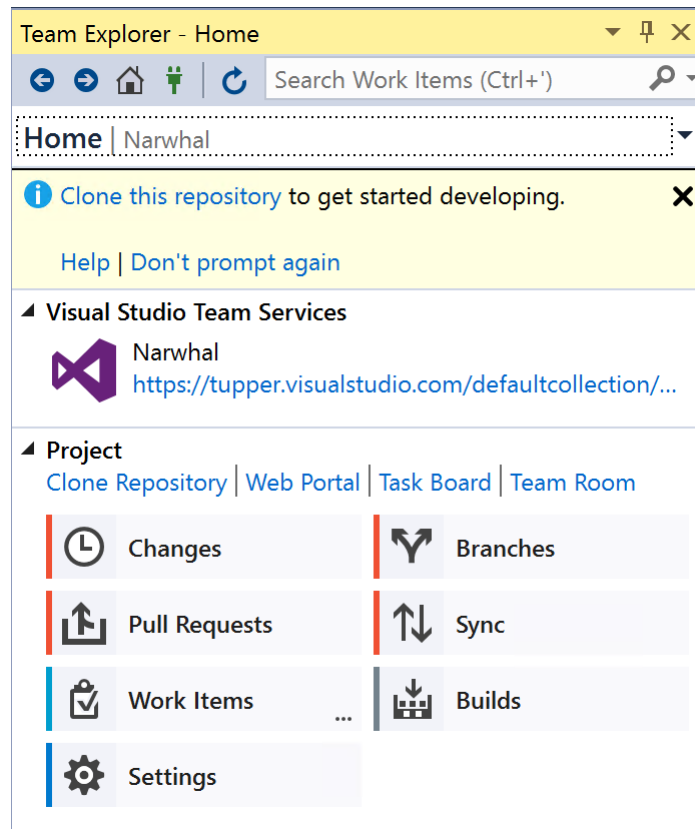


Figure 20: Connected to a Team Project in VSTS