

Table of Contents

[Visual Studio Documentation](#)

[Overview](#)

[Installation](#)

[Install Visual Studio](#)

[Sign in to Visual Studio](#)

[Work with multiple user accounts](#)

[Extend a trial version or update a license](#)

[Install offline](#)

[Create an offline installation of Visual Studio](#)

[Install required certificates for offline installation](#)

[Install on low-bandwidth or unreliable network environments](#)

[Troubleshooting installation issues](#)

[Update Visual Studio](#)

[Modify Visual Studio](#)

[Install Help Viewer](#)

[Repair Visual Studio](#)

[Uninstall Visual Studio](#)

[Visual Studio Administrator Guide](#)

[Use the command-line](#)

[Install on a network](#)

[Deploy in an enterprise](#)

[Manage installations](#)

[Manage subscriptions](#)

[Help Viewer Administrator Guide](#)

[Reference](#)

[Use Visual Studio from an Azure virtual machine](#)

[Install Build Tools into a Container](#)

[Advanced Example for Containers](#)

[Known Issues for Containers](#)

Quickstarts

[Visual Studio Orientation](#)

[Projects and Solutions](#)

[Use the Editor](#)

[Personalize Theme and Text Colors](#)

[C++: Create a Console App](#)

[Python: Create a Web App](#)

[Node.js: Create a Web App](#)

[C#: Create a Web App](#)

[Visual Basic: Create a Console App](#)

Tutorials

[C++ in Visual Studio...](#)

[Python in Visual Studio...](#)

[R in Visual Studio...](#)

[Node.js in Visual Studio](#)

[C# in Visual Studio](#)

[Visual Basic in Visual Studio](#)

[Debugging in Visual Studio](#)

How-to guides

[Develop](#)

[Move Around in the Visual Studio IDE](#)

[Work with Solutions and Projects](#)

[Develop without Projects or Solutions \("Open Folder"\)](#)

[Use the Editor](#)

[Cross-Platform Mobile Development](#)

[Office and Sharepoint Development](#)

[Work with XML and XSLT Files](#)

[Additional Tutorials](#)

[Access Data...](#)

[Design User Interfaces...](#)

[Compile and Build](#)

[Walkthrough: Building an Application](#)

- [Building and Cleaning Projects and Solutions](#)
- [Understanding Build Configurations](#)
- [Understanding Build Platforms](#)
- [MSBuild...](#)
- [VSTS and TFS...](#)
- [Specify Build Events \(Visual Basic\)](#)
- [Specify Build Events \(C#\)](#)
- [Configuring Warnings in Visual Basic](#)
- [Disable the Hosting Process](#)
- [Walkthrough: Creating a Multiple-Computer Build Environment](#)
- [Debug...](#)
- [Test...](#)
- [Measure performance...](#)
- [Analyze code quality...](#)
- [Deploy...](#)
- [Extend Visual Studio...](#)
- [Analyze and Model Architecture...](#)
- [Customize the IDE](#)
 - [Change Fonts and Colors](#)
 - [Customize Menus and Toolbars](#)
 - [Customizing window layouts](#)
 - [Customizing the Start Page](#)
 - [Find and use Visual Studio Extensions](#)
 - [Managing External Tools](#)
 - [Synchronized Settings](#)
- [Optimize Visual Studio Performance](#)
 - [Optimize Startup Time](#)
 - [Optimize Solution Loading](#)
 - [Performance Tips and Tricks](#)
- [Security in Visual Studio](#)
 - [Securing Applications](#)
 - [Maintaining Security](#)

User Permissions and Visual Studio
Security Bibliography
Manage Accessibility Features
Set IDE Accessibility Options
Use the Keyboard Exclusively
Accessibility Tips and Tricks
Accessibility Products and Services from Microsoft
Resources for Designing Accessible Applications
Globalizing and Localizing Applications
Introduction to International Applications Based on the .NET Framework
Localizing Applications
Globalizing Applications
Creating Applications in Bi-directional Languages

Reference

Project and Item Templates
Template Parameters
Template Schema Reference (Extensibility)

General User Interface Elements

Call Hierarchy
Preview Changes
Choose Toolbox Items, WPF Components
Code Snippet Picker
Command Window
Convert Dialog Box
Error List Window
File Properties, JavaScript
Go To Line
Immediate Window
Miscellaneous Files
Options Dialog Box
Output Window
Project Properties Reference

[Property Pages, JavaScript](#)

[Properties Window](#)

[Toolbox](#)

[Devenv Command Line Switches](#)

[-? \(devenv.exe\)](#)

[-Build \(devenv.exe\)](#)

[-Clean \(devenv.exe\)](#)

[-Command \(devenv.exe\)](#)

[-DebugExe \(devenv.exe\)](#)

[-Deploy \(devenv.exe\)](#)

[-Diff](#)

[-Edit \(devenv.exe\)](#)

[-LCID \(devenv.exe\)](#)

[-Log \(devenv.exe\)](#)

[-Out \(devenv.exe\)](#)

[-Project \(devenv.exe\)](#)

[-ProjectConfig \(devenv.exe\)](#)

[-Rebuild \(devenv.exe\)](#)

[-ResetSettings \(devenv.exe\)](#)

[-ResetSkipPkgs \(devenv.exe\)](#)

[-Run \(devenv.exe\)](#)

[-Runexit \(devenv.exe\)](#)

[-SafeMode \(devenv.exe\)](#)

[-Setup \(devenv.exe\)](#)

[-Upgrade \(devenv.exe\)](#)

[-UseEnv \(devenv.exe\)](#)

[Visual Studio Commands](#)

[Visual Studio Command Aliases](#)

[Add Existing Item Command](#)

[Add Existing Project Command](#)

[Add New Item Command](#)

[Alias Command](#)

Evaluate Statement Command
Find Command
Find in Files Command
Go To Command
Import and Export Settings Command
List Call Stack Command
List Disassembly Command
List Memory Command
List Modules Command
List Registers Command
List Source Command
List Threads Command
Log Command Window Output Command
New File Command
Open File Command
Open Project Command
Open Solution Command
Print Command
Quick Watch Command
Replace Command
Replace In Files Command
Set Current Process
Set Current Stack Frame Command
Set Current Thread Command
Set Radix Command
Shell Command
ShowWebBrowser Command
Start Command
Symbol Path Command
Toggle Breakpoint Command
Watch Command
Microsoft Help Viewer

[Install and Manage Local Content](#)

[Finding topics in Help Viewer](#)

[Customize the Help Viewer](#)

[Accessibility Features of the Help Viewer](#)

[Dotfuscator Community Edition \(CE\)](#)

[Capabilities of Dotfuscator](#)

[Install Dotfuscator CE](#)

[Upgrade Dotfuscator CE](#)

Resources

[What's New](#)

[Release notes & system requirements](#)

[Current release notes](#)

[Preview release notes](#)

[Release notes history](#)

[Release rhythm](#)

[System requirements](#)

[Platform compatibility](#)

[Licensing](#)

[Distributable code](#)

[Support lifecycle and servicing](#)

[How to Report a Problem with Visual Studio](#)

[Resources for Troubleshooting IDE Errors](#)

[Talk to Us](#)

Learn how Visual Studio helps you develop apps using a variety of programming languages. Tutorials, videos, and other documentation show you ways to create code and apps by using Visual Studio.



[Learn how to install and set up Visual Studio](#)

[Get an overview of Visual Studio](#)

[Visual Studio 2017 on Microsoft Virtual Academy](#)

Experience Visual Studio with 5-minute "Quickstarts"

[Get oriented with the integrated development environment \(IDE\)](#)

[Learn about projects and solutions](#)

[Use the code editor](#)

[Personalize theme and text colors](#)

[Create a web app with C#](#)

[Create a console app with Visual Basic](#)

[Create a console app with C++](#)

[Create a web app with Python](#)

[Create a web app with Node.js](#)

Go deeper with tutorials

C++

Node.js

Python

R

C#

Visual Basic

Debugging

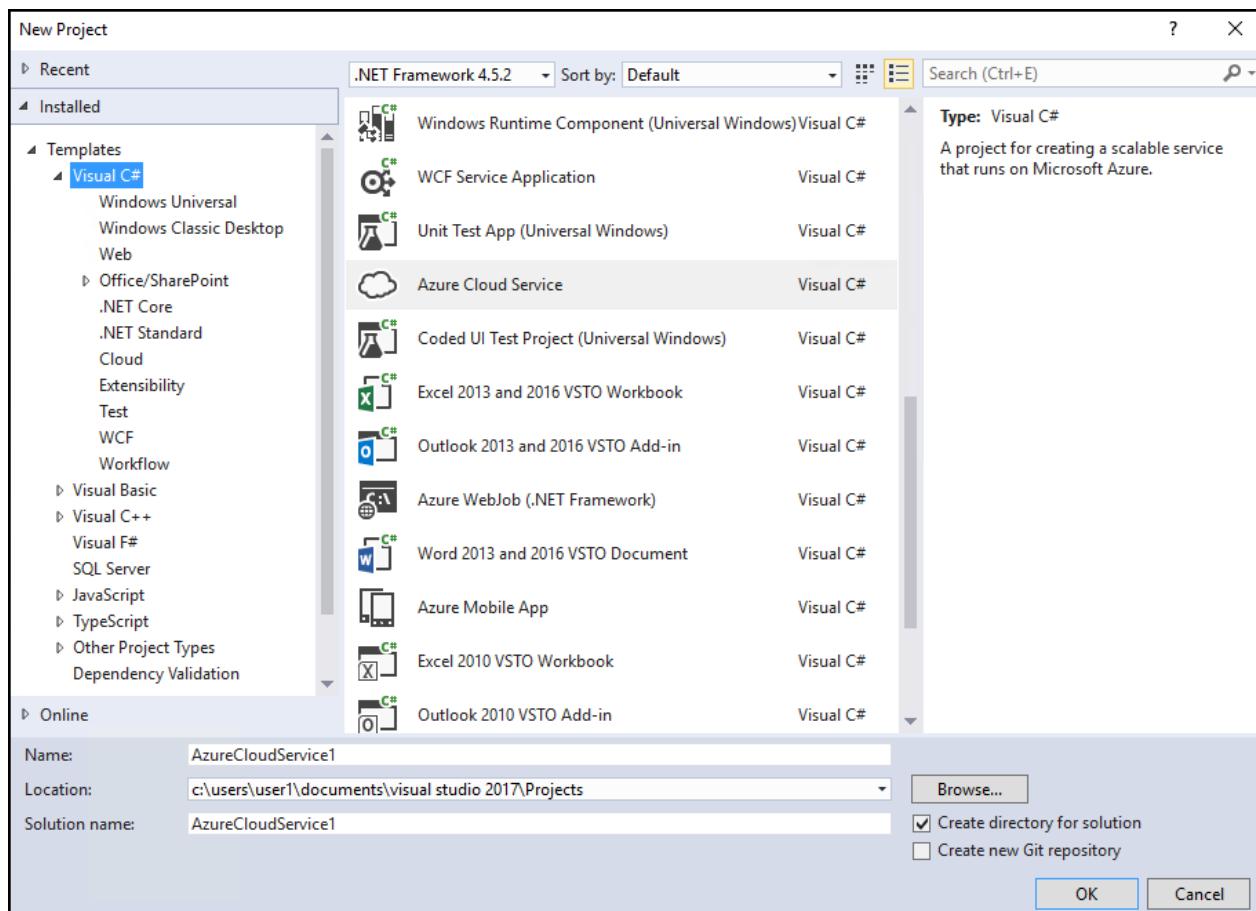
Visual Studio IDE overview

3/15/2018 • 15 min to read • [Edit Online](#)

The Visual Studio interactive development environment (IDE) is a creative launching pad that you can use to view and edit nearly any kind of code, and then debug, build, and publish apps for Android, iOS, Windows, the web, and the cloud. There are versions available for Mac and Windows. This topic introduces you to the features of the Visual Studio IDE. We'll walk through some things you can do with Visual Studio and how to install and use it, create a simple project, get pointers on debugging and deploying code, and take a tour of the various tool windows.

What can you do with the Visual Studio IDE?

Do you want to create an app for an Android phone? You can do that. How about create a cutting edge game using C++? You can do that too and much, much more. Visual Studio provides templates that help you make websites, games, desktop apps, mobile apps, apps for Office, and more.



Or, you can simply open some code you get from almost anywhere and get working. See a project on GitHub that you like? Just clone the repository, open it in Visual Studio, and start coding!

Create mobile apps

You can create native mobile apps for different platforms by using C# and Xamarin, or Visual C++, or hybrid apps using JavaScript with Apache Cordova. You can write mobile games for Unity, Unreal, DirectX, Cocos, and more. Visual Studio includes an Android emulator to help you run and debug Android apps.

You can leverage the power of the cloud for your mobile apps by creating Azure app services. Azure app services enable your apps to store data on the cloud, securely authenticate users, and automatically scale its resources up

or down to accommodate the needs of your app and your business. To learn more, see [Mobile app development](#).

Create cloud apps for Azure

Visual Studio offers a suite of tools that enable you to easily create cloud-enabled applications powered by Microsoft Azure. You can configure, build, debug, package, and deploy applications and services on Microsoft Azure directly from the IDE. To get Azure Tools for .NET, select the **Azure development** workload when you install Visual Studio. For more information, see [Visual Studio Tools for Azure](#).

You can leverage Azure services for your apps using Connected Services such as:

- [Azure Mobile Services](#)
- [Azure Storage](#)

[HockeyApp](#) helps you distribute beta versions, collect live crash reports, and get feedback from real users. In addition, you can integrate Office 365 REST APIs into your own app to connect to data stored in the cloud. For more information, see [these GitHub samples](#).

[Application Insights](#) helps you detect and diagnose quality issues in your apps and web services. Application Insights also helps you understand what your users actually do with your app, so you can optimize the user experience.

Create apps for the web

The web drives our modern world, and Visual Studio can help you write apps for it. You can create web apps using ASP.NET, Node.js, Python, JavaScript and TypeScript. Visual Studio understands web frameworks like Angular, jQuery, Express, and more. ASP.NET Core and .NET Core run on Windows, Mac, and Linux operating systems. [ASP.NET Core](#) is a major update to MVC, WebAPI and SignalR, and runs on Windows, Mac, and Linux. ASP.NET Core has been designed from the ground up to provide you with a lean and composable .NET stack for building modern cloud-based web apps and services.

For more information, see [Modern Web Tooling](#).

Build cross-platform apps and games

You can use Visual Studio to build apps and games for macOS, Linux, and Windows, as well as for Android, iOS, and other mobile devices.

- Build [.NET Core](#) apps that run on Windows, macOS and Linux.
- Build mobile apps for iOS, Android, and Windows in C# and F# by using [Xamarin](#).
- Use standard web technologies—HTML, CSS, and JavaScript—to build mobile apps for iOS, Android, and Windows by using [Apache Cordova](#).
- Build 2D and 3D games in C# by using [Visual Studio Tools for Unity](#).
- Build native C++ apps for iOS, Android and Windows devices, and share common code in libraries built for iOS, Android, and Windows, by using [C++ for cross-platform development](#).
- Deploy, test, and debug Android apps with the [Android emulator](#).

Visual Studio can do help you do many more things. For a more complete list, see www.visualstudio.com.

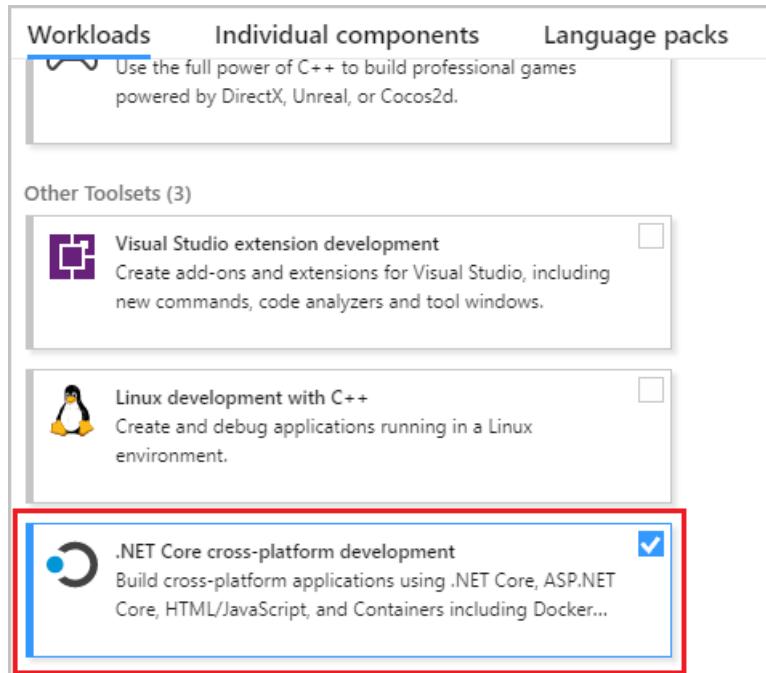
Install the Visual Studio IDE

To get started, download Visual Studio and install it on your system. You can download it at [Visual Studio 2017](#).

Visual Studio is now more lightweight than ever. The modular installer enables you to choose and install *workloads*, which are groups of features needed for the programming language or platform you prefer. This strategy helps keep the footprint of the Visual Studio installation smaller than ever before, which means it installs

and updates faster too. In addition to improved installation performance, Visual Studio 2017 also has shorter IDE start-up and solution load times.

To learn more about setting up Visual Studio on your system, see [Install Visual Studio 2017](#). To follow the steps for [creating a program](#), be sure to select the **.NET Core cross-platform development** workload during the installation.



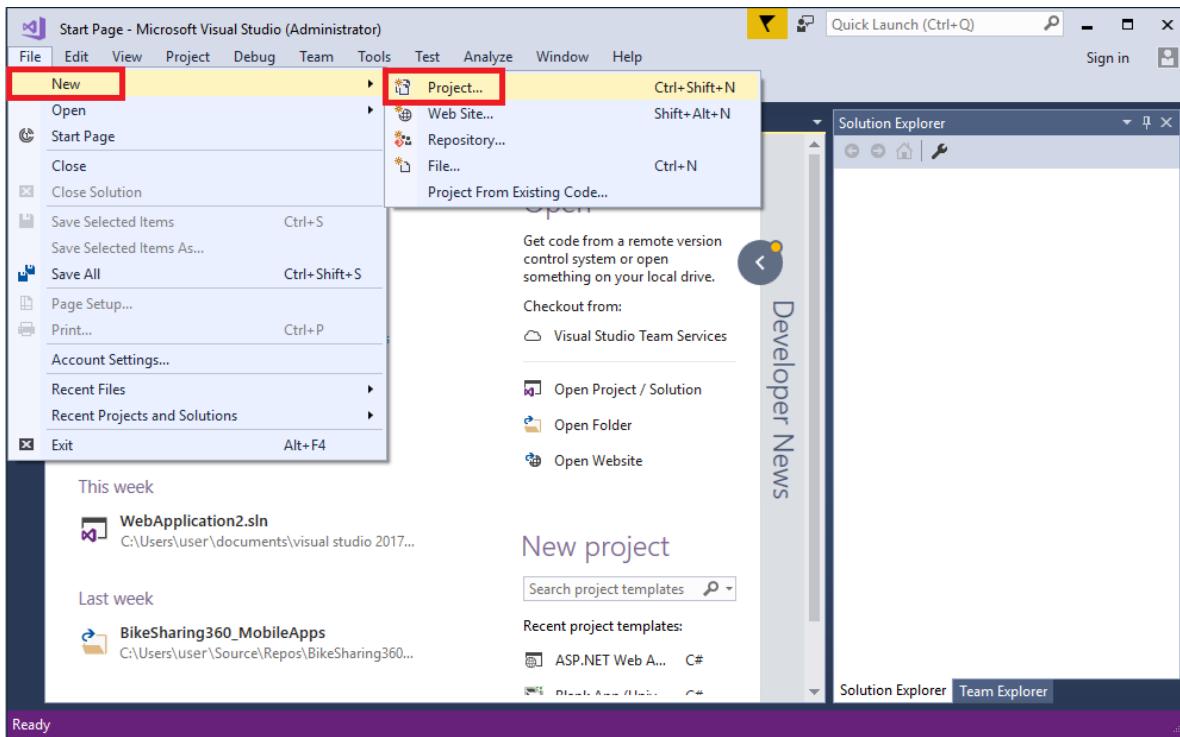
Sign in

When you start Visual Studio for the first time, you can optionally sign in using your Microsoft account, or your work or school account. Being signed in lets you synchronize Visual Studio settings, such as window layouts, across multiple devices. It also connects you automatically to the services you might need, such as Azure subscriptions and [Visual Studio Team Services](#).

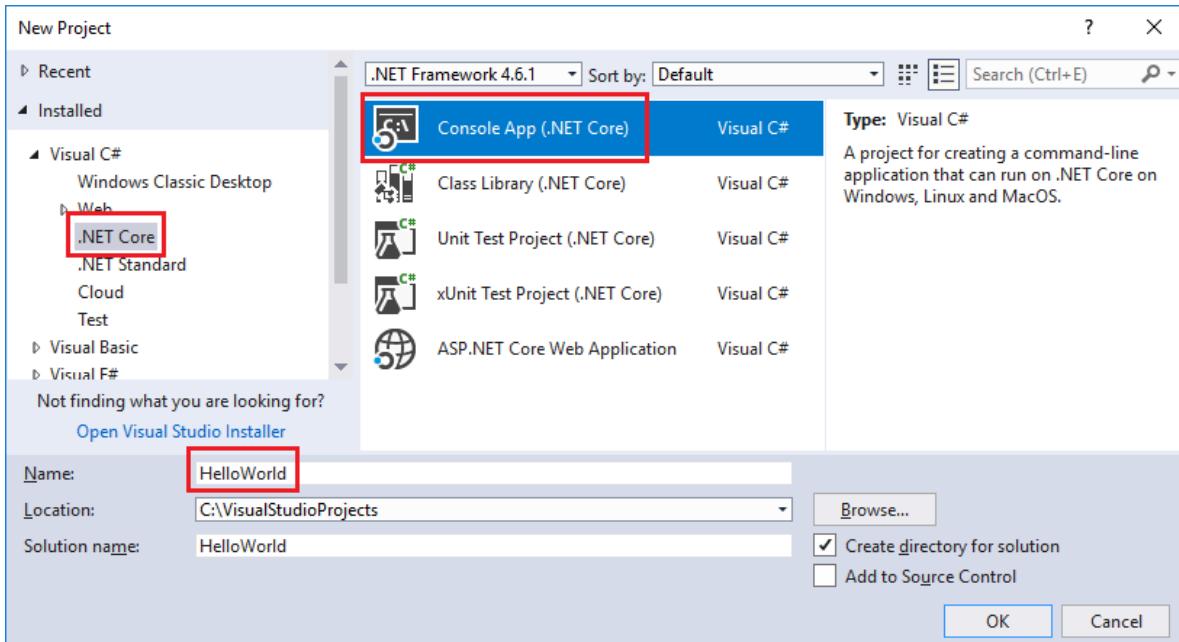
Create a program

One good way to learn about something is to use it! Let's dive in and create a new, simple program.

1. Open Visual Studio. On the menu, choose **File > New > Project....**



2. The **New Project** dialog box shows several project templates. Choose the **.NET Core** category under **Visual C#**, and then choose the **Console App (.NET Core)** template. In the **Name** text box, type "HelloWorld". Select the **OK** button.

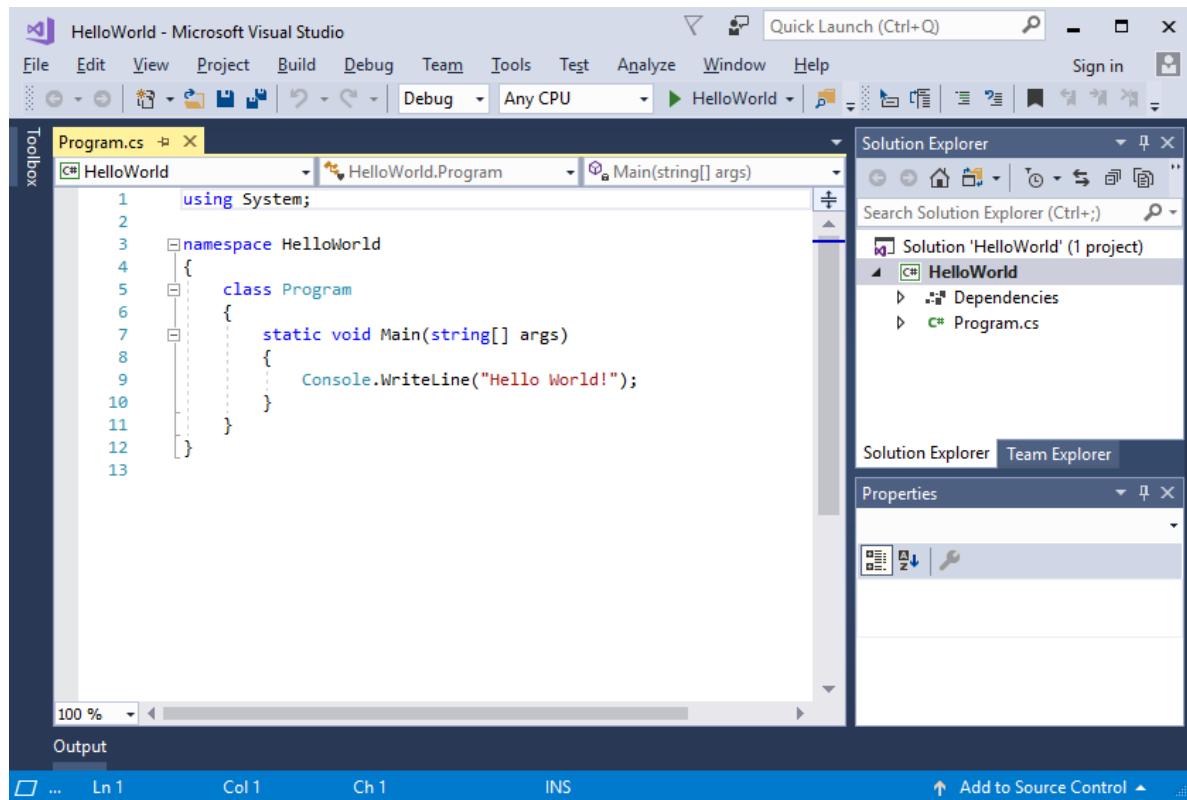


NOTE

If you don't see the **.NET Core** category, you need to install the **.NET Core cross-platform development** workload. To do this, choose the **Open Visual Studio Installer** link on the bottom left of the **New Project** dialog. After **Visual Studio Installer** opens, scroll down and select the **.NET Core cross-platform development** workload, and then choose **Modify**.

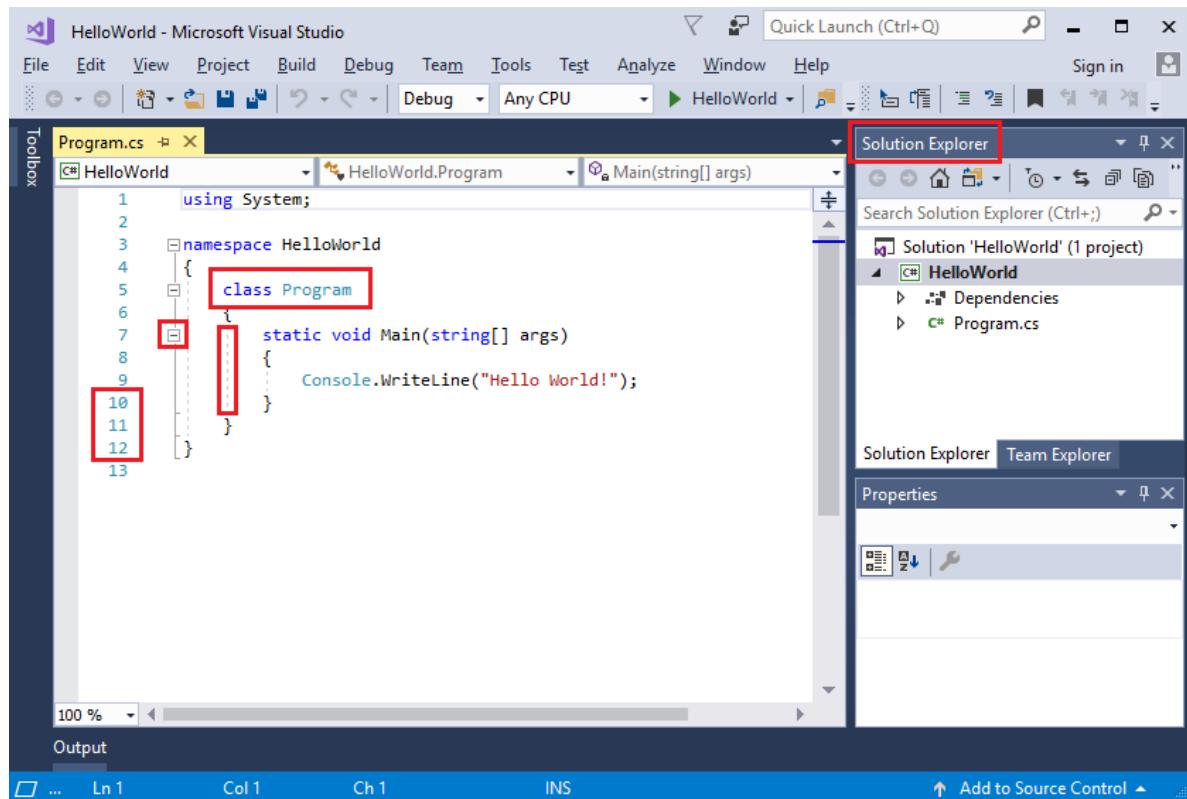
Visual Studio uses the template to create your project. It is a simple "Hello World" application that calls the `WriteLine()` method to display the literal string "Hello World!" in the console window.

3. Shortly, you should see something like the following screenshot:



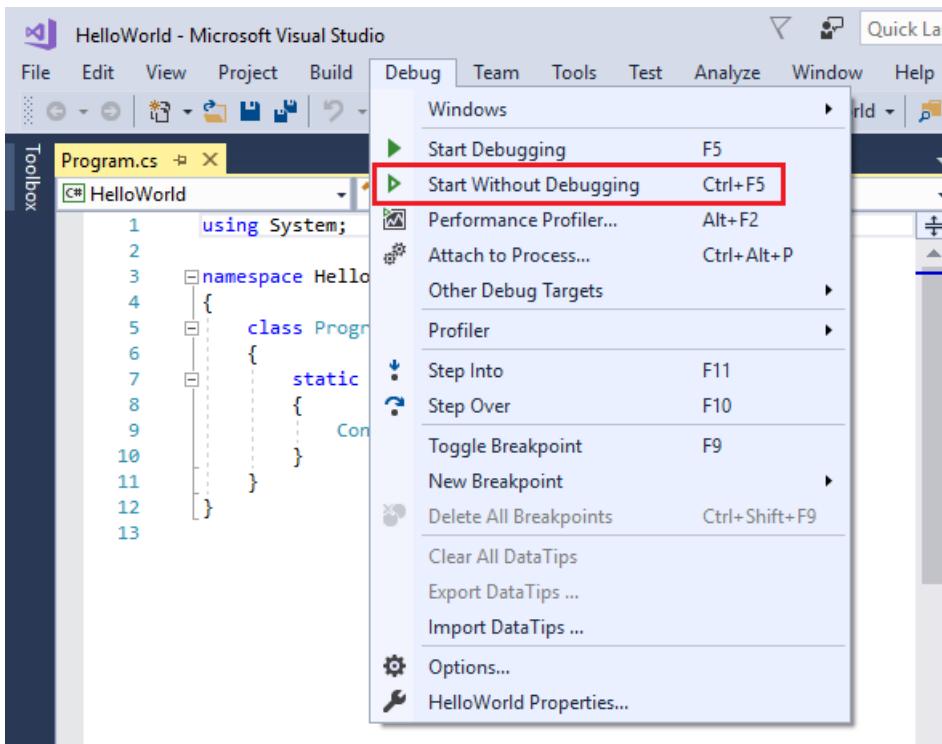
The C# code for your application is shown in the editor window, which takes up most of the space. Notice that the code syntax is automatically colorized to indicate different types of code, such as keywords and types. In addition, small, vertical dashed lines in the code indicate which braces match one another, and line numbers help you locate code later. You can choose the small, boxed minus signs to collapse or expand code. This code outlining feature lets you hide code you don't need, helping to minimize onscreen clutter.

The project files are listed on the right side in a window called **Solution Explorer**.

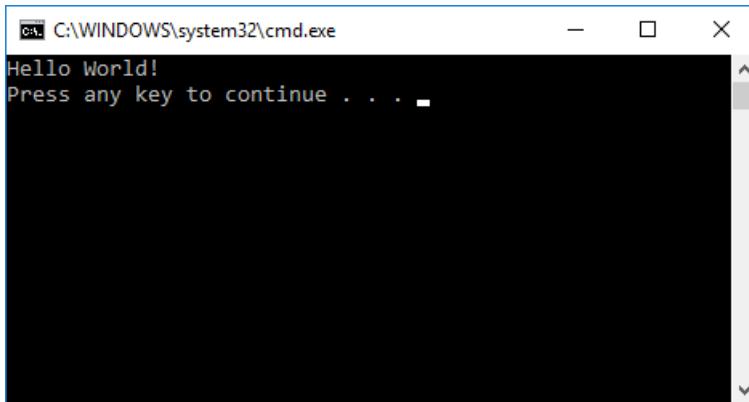


There are other menus and tool windows available, but let's move on for now.

4. Now, start the app. You can do this by choosing **Start Without Debugging** from the **Debug** menu on the menu bar. You can also press **Ctrl+F5**.



Visual Studio builds the app, and a console window opens with the message "Hello World!". You now have a running app!



5. To close the console window, press any key on your keyboard.
6. Let's add some additional code to the app. Add the following C# code before the line that says

```
Console.WriteLine("Hello World!");
```

```
Console.WriteLine("\nWhat is your name?");
var name = Console.ReadLine();
```

This code displays "What is your name?" in the console window, and waits until the user enters some text followed by the **Enter** key.

7. Now change the line that says `Console.WriteLine("Hello World!");` to the following code:

```
Console.WriteLine($"\\nHello {name}!");
```

8. Run the app again by selecting **Debug > Start Without Debugging** or by pressing **Ctrl+F5**.

Visual Studio rebuilds the app, and a console window opens and prompts you for your name.

9. Enter your name in the console window and press **Enter**.

The screenshot shows a Windows Command Prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The window contains the following text:
What is your name?
Genevieve
Hello Genevieve!
Press any key to continue . . .

10. Press any key to close the console window.

Debug, test, and improve your code

Nothing runs perfectly all the time. When you write code, you need to run it and test it for bugs and performance. Visual Studio's cutting edge debugging system enables you to debug code running in your local project, on a remote device, or on an emulator such as [the one for Android devices](#). You can step through code one statement at a time and inspect variables as you go. You can set breakpoints that are only hit when a specified condition is true. You can monitor the values of variables as the code runs, and more. All of this can be managed in the code editor itself, so that you don't have to leave your code. To get more details about debugging in Visual Studio, see [Debugger feature tour](#).

For testing, Visual Studio offers unit testing, IntelliTest, load and performance testing, and more. To learn more about testing, see [Testing tools and scenarios](#). To learn more about improving the performance of your apps, see [Profiling feature tour](#).

Deploy your finished application

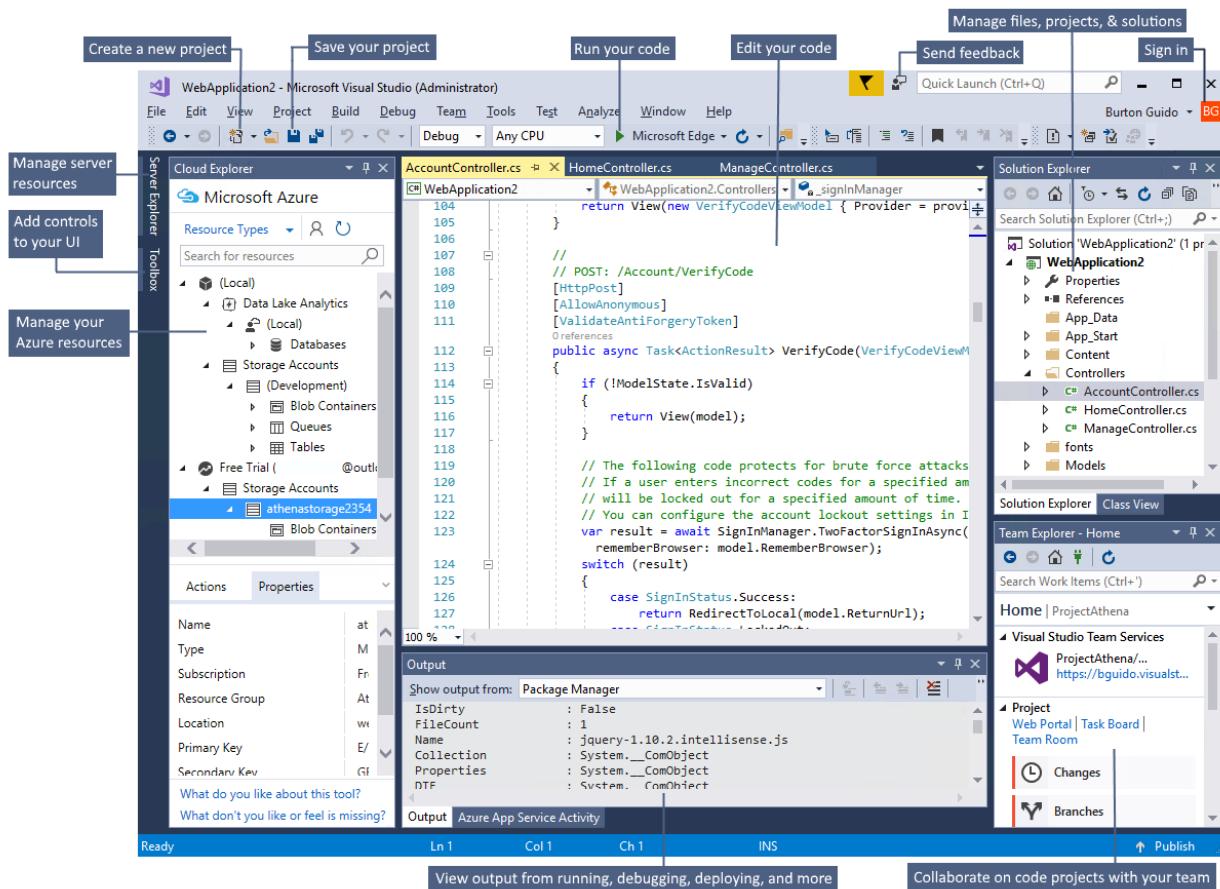
When your application is ready to deploy to users or customers, Visual Studio provides the tools to do that, whether you're deploying to Microsoft Store, to a SharePoint site, or with InstallShield or Windows Installer technologies. It's all accessible through the IDE. For more information, see [Deploying applications, services, and components](#).

Quick tour of the IDE

To give you a high-level visual overview of Visual Studio, the following image shows Visual Studio with an open project along with several key tool windows you will most likely use:

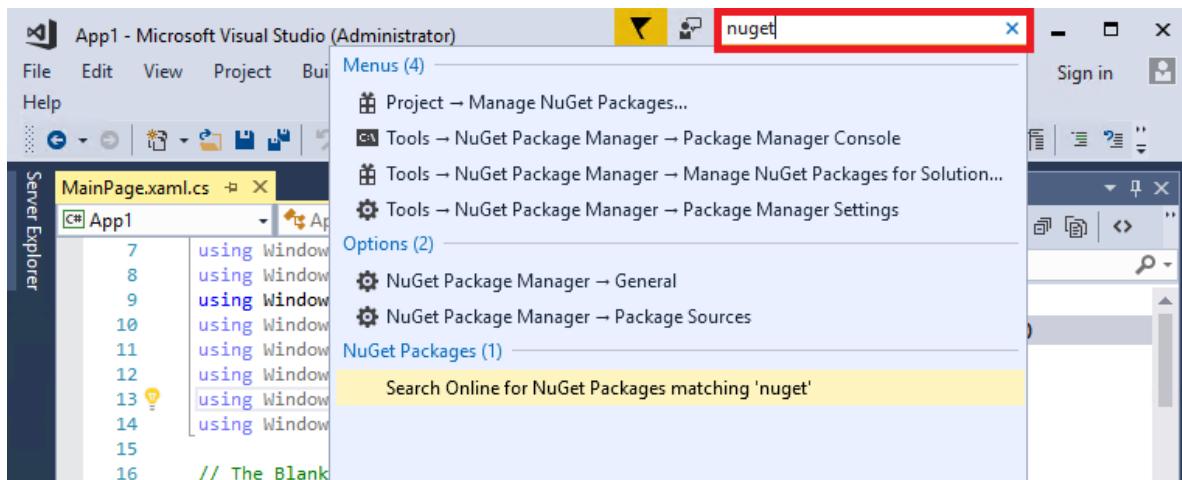
- [Solution Explorer](#) lets you view, navigate, and manage your code files. Solution Explorer can help organize your code by grouping the files into solutions and projects.
- The [Editor](#) window, where you'll likely spend a majority of your time, shows your code and enables you to edit source code and design a UI.
- The [Output](#) window is where Visual Studio sends its notifications, such as debugging and error messages, compiler warnings, publishing status messages, and more. Each message source has its own tab.
- [Team Explorer \(VSTS\)](#) lets you track work items and share code with others using version control technologies such as [Git](#) and [Team Foundation Version Control \(TFVC\)](#).
- [Cloud Explorer](#) lets you view and manage your Azure resources, such as virtual machines, tables, SQL databases, and more. If a particular operation requires the Azure portal, Cloud Explorer provides links that

take you to the place in the Azure portal you need to go.

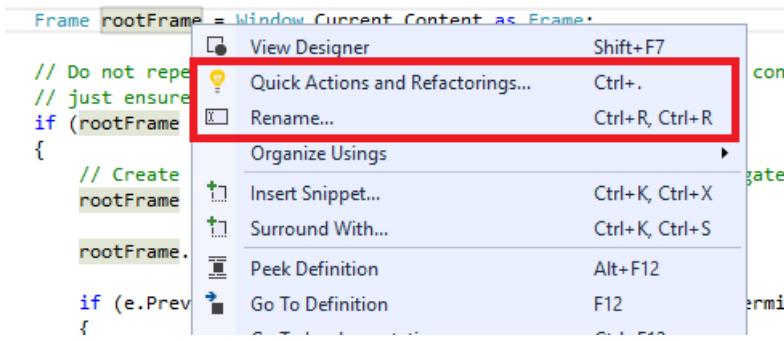


Following are some other common productivity features in Visual Studio:

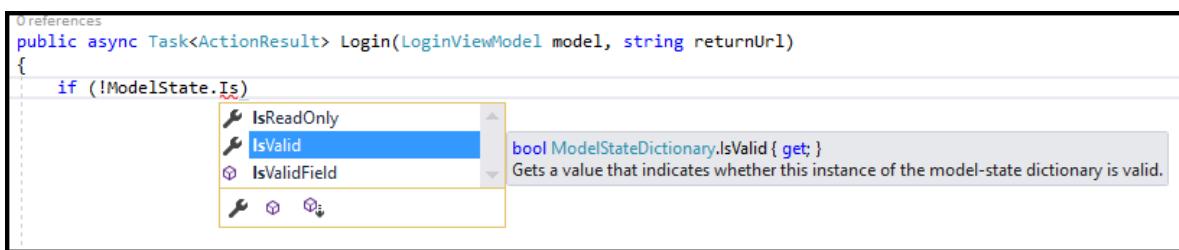
- The [Quick Launch](#) search box is a great way to rapidly find what you need in Visual Studio. Just start entering in the name of whatever you are looking for, and Visual Studio lists results that take you exactly where you want to go. Quick Launch also shows links that start the Visual Studio Installer for any workload or individual component.



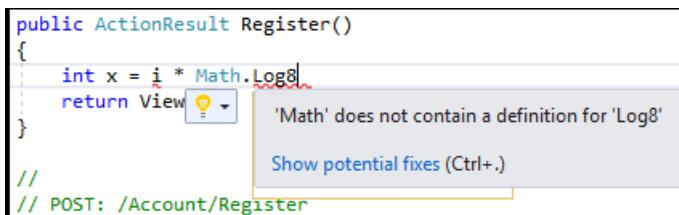
- [Refactoring](#) includes operations such as intelligent renaming of variables, moving selected lines of code into a separate function, moving code to other locations, reordering function parameters, and more.



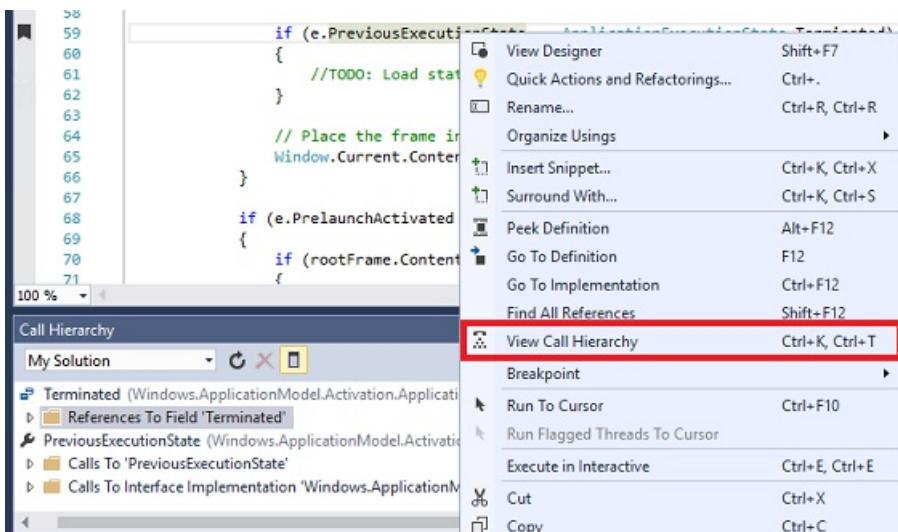
- **IntelliSense** is an umbrella term for a set of popular features that display type information about your code directly in the editor and, in some cases, write small bits of code for you. It's like having basic documentation inline in the editor, which saves you from having to look up type information in a separate help window. IntelliSense features vary by language. For more information, see [C# IntelliSense](#), [Visual C++ IntelliSense](#), [JavaScript IntelliSense](#), and [Visual Basic IntelliSense](#). The following illustration shows some IntelliSense features at work:



- **Squiggles** are wavy red underlines that alert you to errors or potential problems in your code in real time as you type. This enables you to fix them immediately without waiting for the error to be discovered during compilation or run time. If you hover over the squiggle, you see additional information about the error. A light bulb may also appear in the left margin with suggestions for how to fix the error. For more information, see [Quick Actions](#).

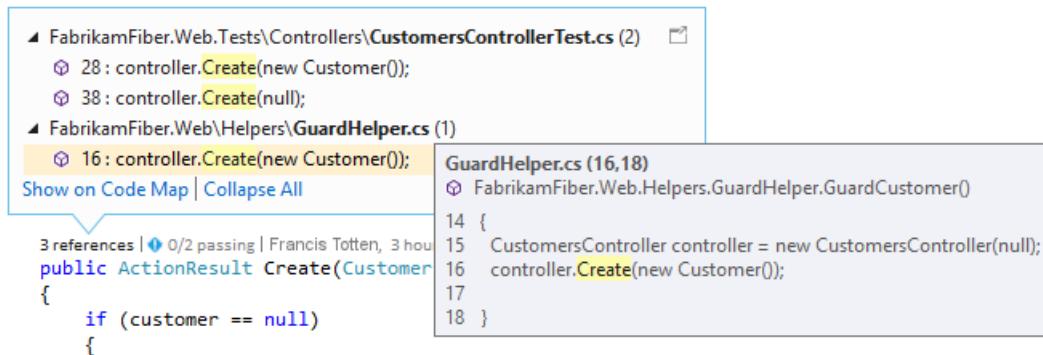


- The [Call Hierarchy](#) window can be opened on the text editor context menu to show the methods that call, and are called by, the method under the caret (insertion point).



- [CodeLens](#) enables you to find references and changes to your code, linked bugs, work items, code reviews,

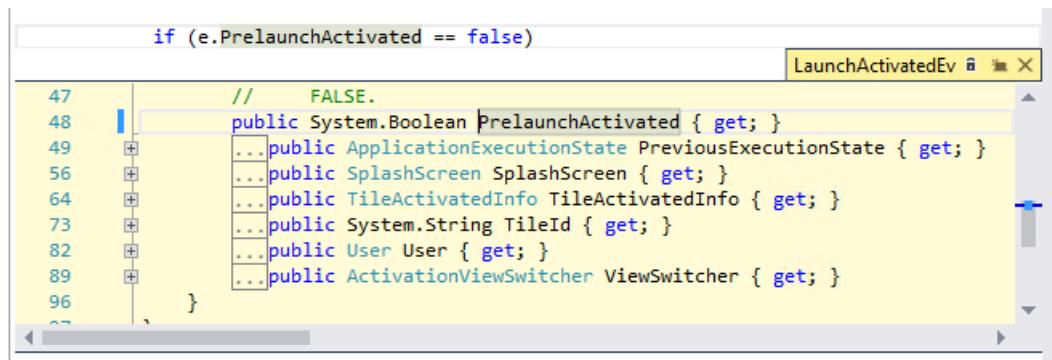
and unit tests, all without leaving the editor.



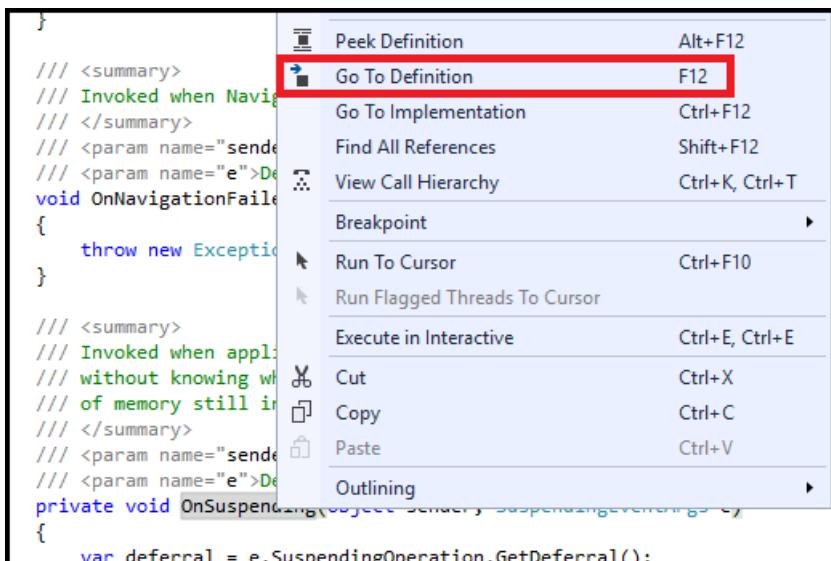
The screenshot shows the Peek to Definition window for the `Create` method in `GuardHelper.cs`. The window displays the code for `GuardCustomer` and its implementation in `CustomersController`. The code is as follows:

```
14 {
15   CustomersController controller = new CustomersController(null);
16   controller.Create(new Customer());
17 }
18 }
```

- The **Peek to Definition** window shows a method or type definition inline, without navigating away from your current context.



- The **Go To Definition** context menu option takes you directly to the place where the function or object is defined. Other navigation commands are also available by right-clicking in the editor.



Manage your source code and collaborate with others

You can manage your source code in Git repos hosted by any provider, including GitHub. Or use [Visual Studio Team Services \(VSTS\)](#) to manage code alongside bugs and work items for your whole project. See [Get Started with Git and Team Services \(VSTS\)](#) to learn more about managing Git repos in Visual Studio using Team Explorer. Visual Studio also has other built-in source control features. To learn more about them, see [New Git Features in Visual Studio 2017 \(blog\)](#).

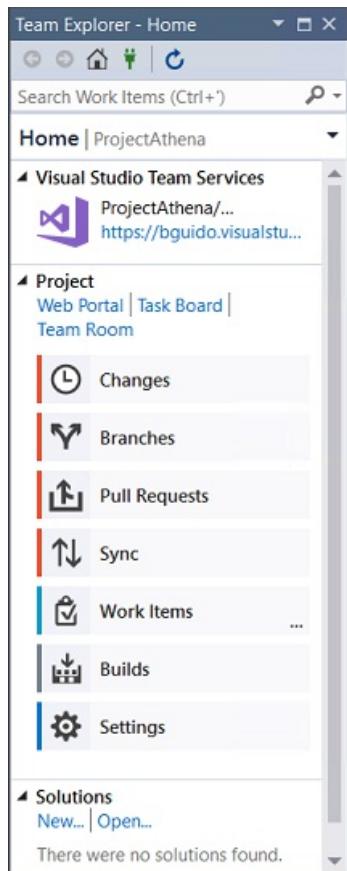
Visual Studio Team Services is a cloud-based service for hosting software projects and enabling collaboration in teams. VSTS supports both Git and Team Foundation Source Control systems, as well as Scrum, CMMI and Agile development methodologies. Team Foundation Version Control (TFVC) uses a single, centralized server repository

to track and version files. Local changes are always checked in to the central server where other developers can get the latest changes.

Team Foundation Server (TFS) is the application lifecycle management hub for Visual Studio. It enables everyone involved with the development process to participate using a single solution. TFS is useful for managing heterogeneous teams and projects, too.

If you have a Visual Studio Team Services account or a Team Foundation Server on your network, you connect to it through the Team Explorer window in Visual Studio. From this window you can check code into or out of source control, manage work items, start builds, and access team rooms and workspaces. You can open Team Explorer from the **Quick Launch** box, or on the main menu from **View, Team Explorer** or from **Team, Manage Connections**.

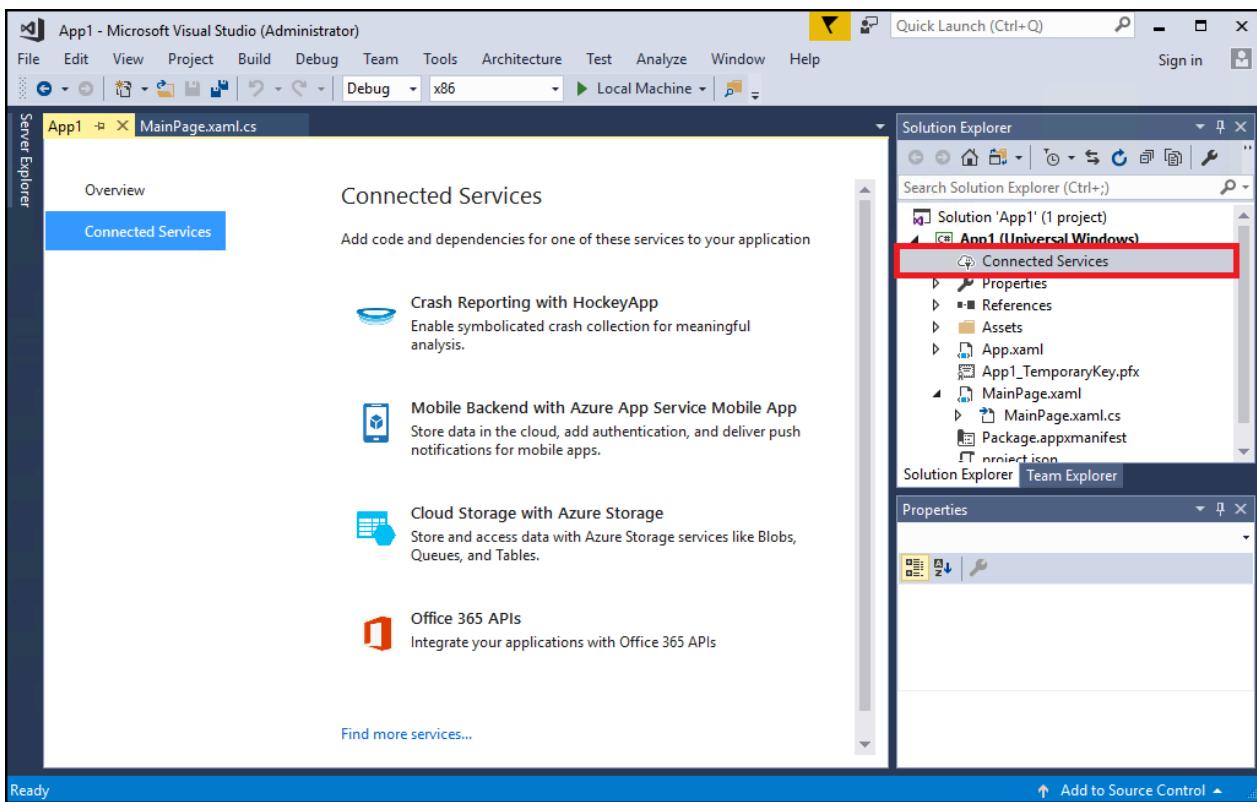
The following image shows the Team Explorer window for a solution that is hosted in VSTS.



You can also automate your build process to build the code that the devs on your team have checked into version control. For example, you can build one or more projects nightly or every time that code is checked in. For more information, see [Build and release \(VSTS and TFS\)](#).

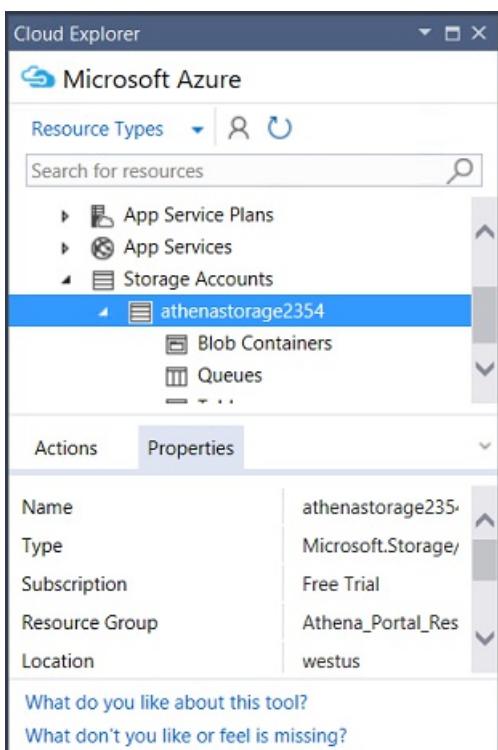
Connect to services, databases, and cloud-based resources

The cloud is critical for today's online world, and Visual Studio provides you the means to leverage it. For example, the Connected Services feature enables you to connect your app to services. Your apps can use it to store their data on Azure storage, among other things.



Choosing a service on the **Connected Services** page starts a Connected Services Wizard that configures your project and downloads the necessary NuGet packages to help get you started coding against the service.

You can view and manage your Azure-based cloud resources within Visual Studio using [Cloud Explorer](#). Cloud Explorer shows the Azure resources in all the accounts managed under the Azure subscription you are logged into. You can get Cloud Explorer by selecting the **Azure development** workload in the Visual Studio installer.

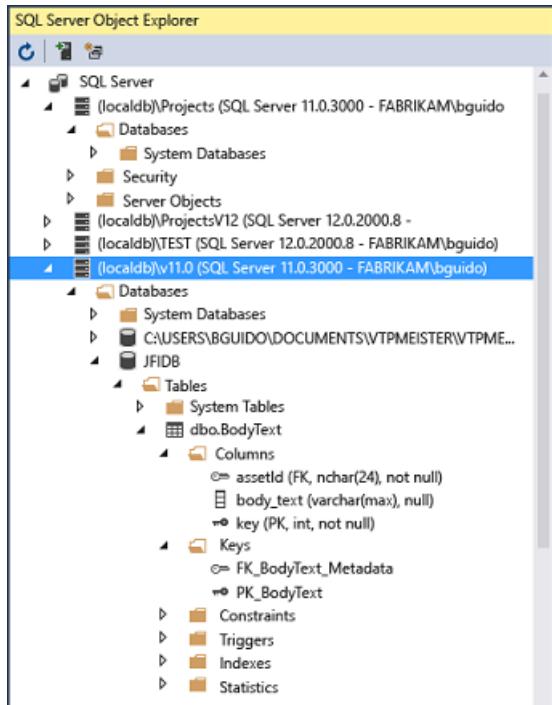


Server Explorer helps you browse and manage SQL Server instances and assets locally, remotely, and on Azure, Salesforce.com, Office 365, and websites. To open Server Explorer, on the main menu, choose **View > Server Explorer**. See [Add new connections](#) for more information on using Server Explorer.

[SQL Server Data Tools \(SSDT\)](#) is a powerful development environment for SQL Server, Azure SQL Database and Azure SQL Data Warehouse. It enables you to build, debug, maintain, and refactor databases. You can work with a

database project, or directly with a connected database instance on- or off-premises.

SQL Server Object Explorer in Visual Studio provides a view of your database objects similar to SQL Server Management Studio. SQL Server Object Explorer enables you to do light-duty database administration and design work, including editing table data, comparing schemas, executing queries by using contextual menus right from SQL Server Object Explorer, and more.



Extend Visual Studio

If Visual Studio doesn't have the exact functionality you need, you can add it! You can personalize the IDE based on your workflow and style, add support for external tools not yet integrated with Visual Studio, and modify existing functionality to increase your productivity. To find the latest version of the Visual Studio Extensibility Tools (VS SDK), see [Visual Studio SDK](#).

You can use the .NET Compiler Platform ("Roslyn") to write your own code analyzers and code generators. Find everything you need at [Roslyn](#).

Find [existing extensions](#) for Visual Studio created by Microsoft developers as well as our development community.

To learn more about extending Visual Studio, see [Extend Visual Studio IDE](#).

Learn more and find out what's new

If you've never used Visual Studio before, look at [Get Started Developing with Visual Studio](#), or check out the free Visual Studio courses available on [Microsoft Virtual Academy](#). If you want to check out the new features in Visual Studio 2017, see [What's New in Visual Studio 2017](#).

Congratulations on completing the tour of the Visual Studio IDE! We hope you learned something useful about some of its main features.

See also

- [Visual Studio IDE](#)
- [Visual Studio downloads](#)
- [The Visual Studio blog](#)

- [Visual Studio forums](#)
- [Microsoft Virtual Academy](#)

Install Visual Studio 2017

3/27/2018 • 5 min to read • [Edit Online](#)

Welcome to a new way to install Visual Studio! In our newest version, we've made it easier for you to select and install just the features you need. We've also reduced the minimum footprint of Visual Studio so that it installs more quickly and with less system impact than ever before.

Want to know more about what else is new in this version? See our [release notes](#).

Ready to install? We'll walk you through it, step-by-step.

Step 1 - Make sure your computer is ready for Visual Studio

Before you begin installing Visual Studio:

1. Check the [system requirements](#). These requirements help you know whether your computer supports Visual Studio 2017.
2. Apply the latest Windows updates. These updates ensure that your computer has both the latest security updates and the required system components for Visual Studio.
3. Reboot. The reboot ensures that any pending installs or updates don't hinder the Visual Studio install.
4. Free up space. Remove unneeded files and applications from your %SystemDrive% by, for example, running the Disk Cleanup app.

For questions about running previous versions of Visual Studio side by side with Visual Studio 2017, see the [Visual Studio compatibility details](#).

Step 2 - Download Visual Studio

Next, download the Visual Studio bootstrapper file. To do so, click the following button, select the edition of Visual Studio 2017 that you want, click **Save**, and then click **Open folder**.



[Watch a video](#) on how to download the Visual Studio bootstrapper file and select the edition of Visual Studio that's right for you.

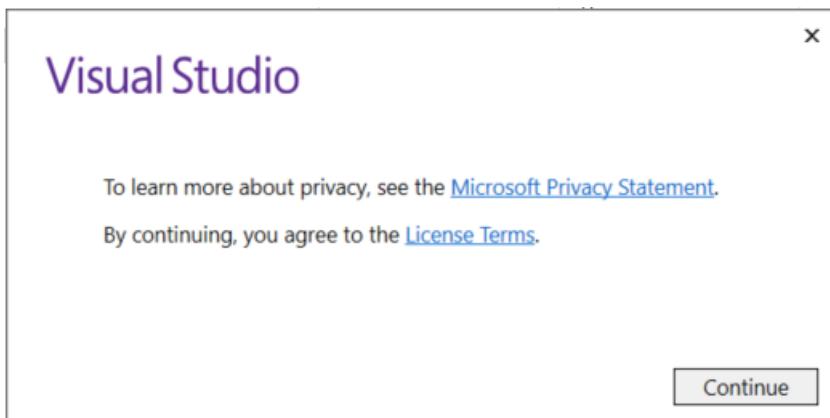
Step 3 - Install the Visual Studio installer

Then, run the bootstrapper file to install the Visual Studio Installer. This new lightweight installer includes everything you need to both install and customize Visual Studio 2017.

1. From your **Downloads** folder, double-click the bootstrapper that matches or is similar to one of the following files:
 - **vs_enterprise.exe** for Visual Studio Enterprise
 - **vs_professional.exe** for Visual Studio Professional
 - **vs_community.exe** for Visual Studio Community

If you receive a User Account Control notice, click **Yes**.

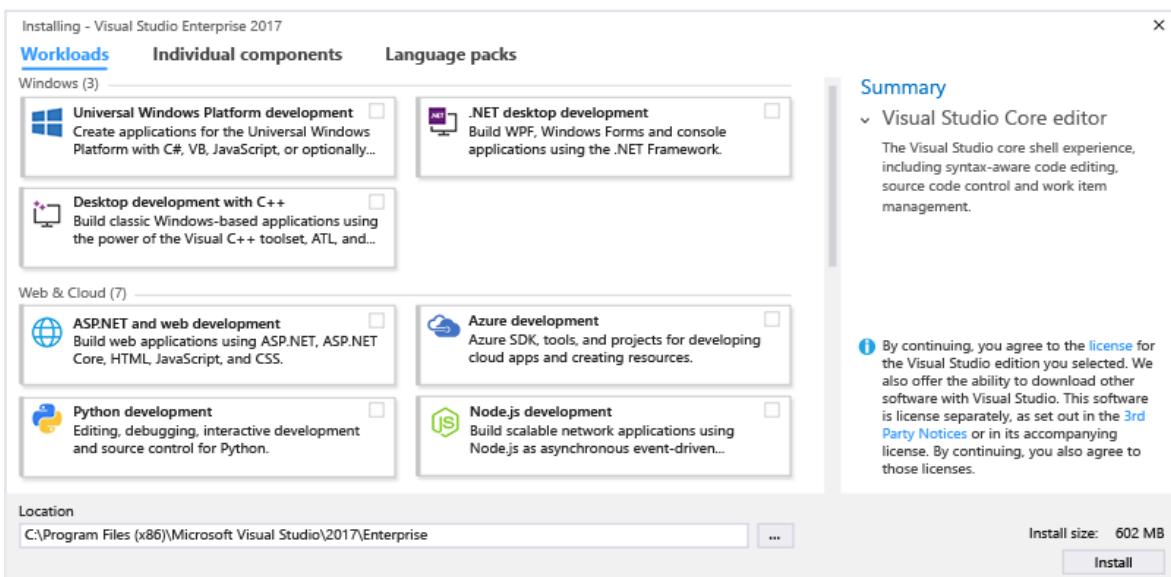
2. We'll ask you to acknowledge the Microsoft [License Terms](#) and the Microsoft [Privacy Statement](#). Click **Continue**.



Step 4 - Select workloads

After the installer is installed, you can use it to customize your installation by selecting the feature sets—or workloads—that you want. Here's how.

1. Find the workload you want in the **Installing Visual Studio** screen.



For example, choose the ".NET desktop development" workload. It comes with the default core editor, which includes basic code editing support for over 20 languages, the ability to open and edit code from any folder without requiring a project, and integrated source code control.

2. After you select the workload(s) you want, click **Install**.

Next, status screens appear that show the progress of your Visual Studio installation.

3. After the new workloads and components are installed, click **Launch**.

TIP

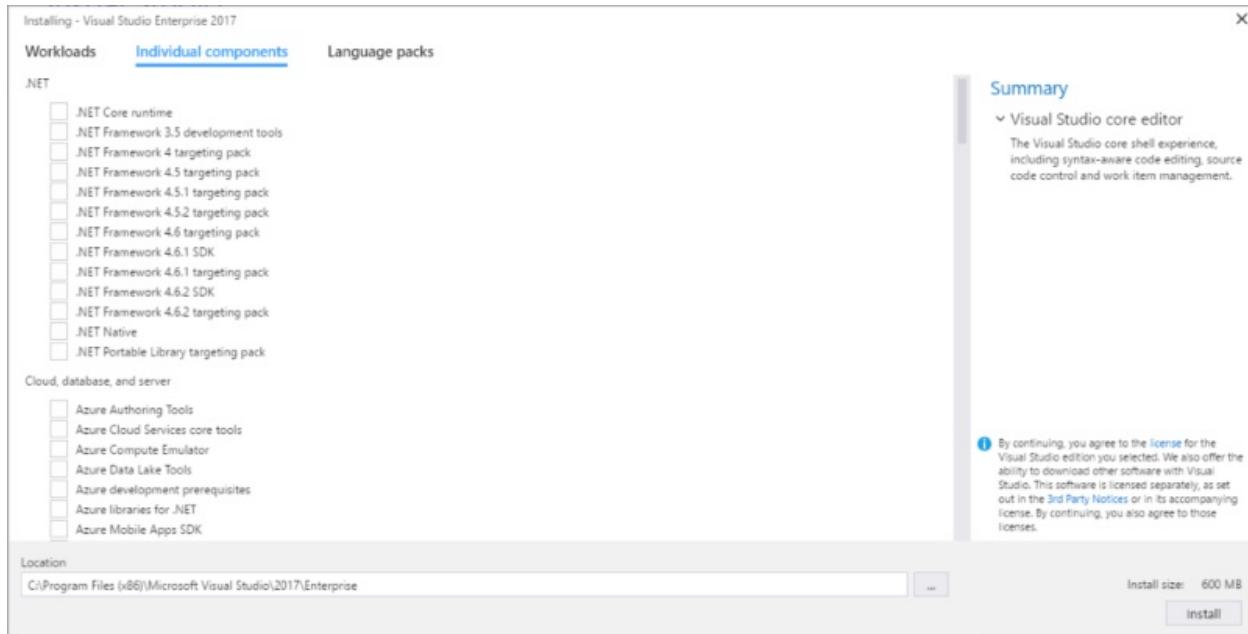
At any time after installation, you can install workloads or components that you didn't install initially. If you have Visual Studio open, go to **Tools > Get Tools and Features...** which opens the Visual Studio Installer. Or, open **Visual Studio Installer** from the Start menu. From there, you can select the workloads or components that you wish to install, then click **Modify**.



[Watch a video](#) on how to install the Visual Studio Installer and then install a workload.

Step 5 - Select individual components (Optional)

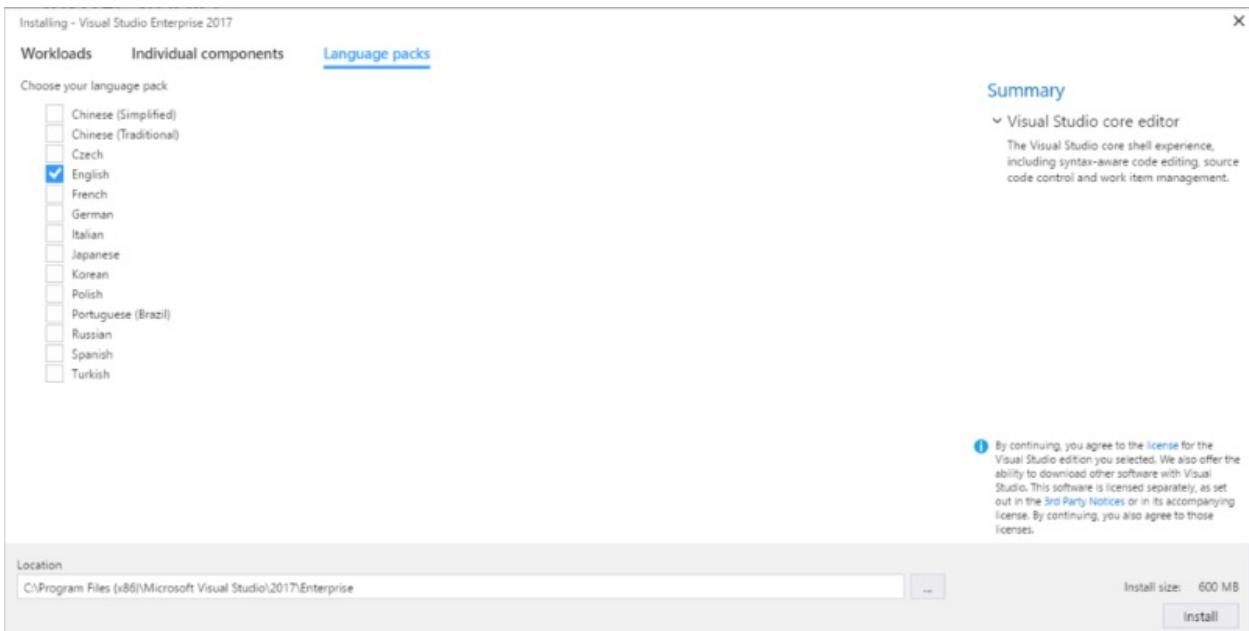
If you don't want to use the Workloads feature to customize your Visual Studio installation, you can do so by installing individual components instead. To select individual components, click the **Individual components** option from the Visual Studio Installer, select what you want, and then follow the prompts.



[Watch a video](#) on how to install an individual component by using the Visual Studio Installer.

Step 6 - Install language packs (Optional)

By default, the installer program tries to match the language of the operating system when it runs for the first time. To install Visual Studio 2017 in a language of your choosing, click the **Language packs** option from the Visual Studio Installer, and follow the prompts.



[Watch a video](#) on how to install a language pack by using the Visual Studio Installer.

Change the installer language from the command line

Another way that you can change the default language is by running the installer from the command line. For example, you can force the installer to run in English by using the following command:

`vs_installer.exe --locale en-US`. The installer will remember this setting when it is run the next time. The installer supports the following language tokens: zh-cn, zh-tw, cs-cz, en-us, es-es, fr-fr, de-de, it-it, ja-jp, ko-kr, pl-pl, pt-br, ru-ru, and tr-tr.

Step 7 - Start developing

1. After Visual Studio installation is complete, click the **Launch** button to [get started developing with Visual Studio](#).
2. Click **File**, and then click **New Project**.
3. Select a project type.

For example, to [build a C++ app](#), click **Installed**, expand **Visual C++**, and then select the C++ project type that you want to build.

To [build a C# app](#), click **Installed**, expand **Visual C#**, and then select the C# project type that you want to build.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).

- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Update Visual Studio 2017](#)
- [Modify Visual Studio 2017](#)
- [Uninstall Visual Studio 2017](#)
- [Create an offline installation of Visual Studio 2017](#)
- [Visual Studio 2017 Administrator Guide](#)
 - [Use command-line parameters to install Visual Studio 2017](#)
- [Install Build Tools into a Container](#)

Sign in to Visual Studio

3/21/2018 • 2 min to read • [Edit Online](#)

You can personalize and optimize your development experience in Visual Studio if you set your Personalization account by signing in to the IDE.

Why should I sign in to Visual Studio?

When you sign in, you enrich your Visual Studio experience. For example, after you sign in, you can synchronize your settings across devices, extend a trial, and automatically connect to an Azure service, to name a few.

Here's a full list of what you can expect and what you can do after you sign in:

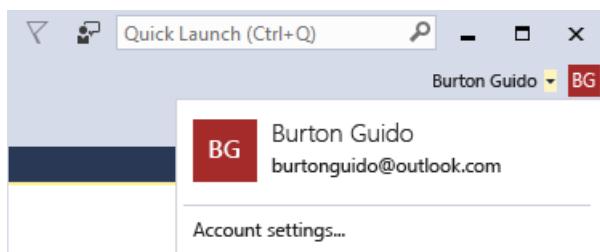
- **Access to the Visual Studio Dev Essentials program** - This program includes free software offerings, training, support, and more. See [Visual Studio Dev Essentials](#) for more information.
- **Synchronize your Visual Studio settings** - Settings that you customize, such as key bindings, window layout, and color theme, apply immediately when you sign in to Visual Studio on any device. See [Synchronized Settings in Visual Studio](#).
- **Unlock the Visual Studio Community edition** - If your Community edition installation prompts you for a license, sign in to the IDE to unlock yourself.
- **Extend the Visual Studio trial period** - You can use Visual Studio Professional or Visual Studio Enterprise for an additional 90 days, instead of being limited to the trial period of 30 days.
- **Unlock Visual Studio if you use an account that's associated with an MSDN or Visual Studio Team Services subscription**. See [How to Unlock Visual Studio](#).
- **Automatically connect to services such as Azure and Visual Studio Team Services** in the IDE without prompting again for credentials for the same account.

How to sign in to Visual Studio

When you start Visual Studio for the first time, you're asked to sign in and provide some basic registration information. You should choose a Microsoft account or a work or school account that best represents you. If you don't have any of these accounts, you can create a Microsoft account for free. See [How do I sign up for a Microsoft account?](#)

Next, choose the UI settings and color theme that you want to use in Visual Studio. Visual Studio remembers these settings and synchronizes them across all Visual Studio environments you have signed in to. For a list of the settings that are synchronized, see [Synchronized Settings](#). You can change the settings later if you open the **Tools**, **Options** menu in Visual Studio.

After you provide the settings, Visual Studio starts, and you're signed in and ready to get started. To verify whether you're signed in, look for your name in the upper-right corner of the Visual Studio environment.



Unless you sign out, you're automatically signed in to Visual Studio whenever you start it, and any changes to synchronized settings are automatically applied. To sign out, choose the down arrow next to your profile name in the upper-right corner of the Visual Studio environment, choose the **Account settings** command, and then choose the **Sign out** link. To sign in again, choose the **Sign in** command in the upper-right corner of the Visual Studio environment.

To change your profile information

1. Go to **File, Account Settings** and choose the **Manage Visual Studio profile** link.
2. In the browser window, choose **Edit profile** and change the settings that you want.
3. When you're done, choose **Save changes**.

Troubleshooting

If you encounter any problems while signing in, please see the [Accounts support page](#) to get help.

See also

[How to Unlock Visual Studio](#)

[Visual Studio IDE Overview](#)

Work with multiple user accounts

3/21/2018 • 4 min to read • [Edit Online](#)

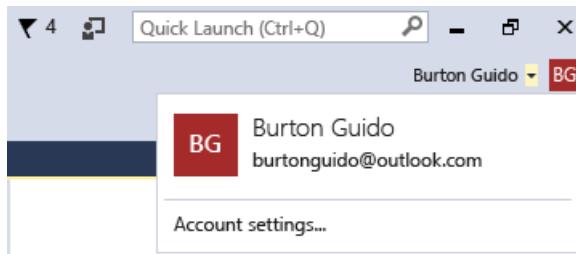
If you have multiple Microsoft accounts and/or work or school accounts, you can add them all to Visual Studio so that you can access the resources from any account without having to sign in to it separately. Currently, Azure, Application Insights, Team Foundation Server, and Office 365 services support the streamlined sign-in experience. Additional services may become available as time goes by.

After you add multiple accounts on one machine, that set of accounts will roam with you if you sign in to Visual Studio on another machine. It is important to note that, although the account names roam, the credentials do not. Therefore, you will be prompted to enter credentials for those other accounts the first time you attempt to use their resources on the new machine.

This walkthrough shows how to add multiple accounts to Visual Studio, and how to see that the resources accessible from those accounts are reflected in places such as the **Add Connected Service** dialog, **Server Explorer**, and **Team Explorer**.

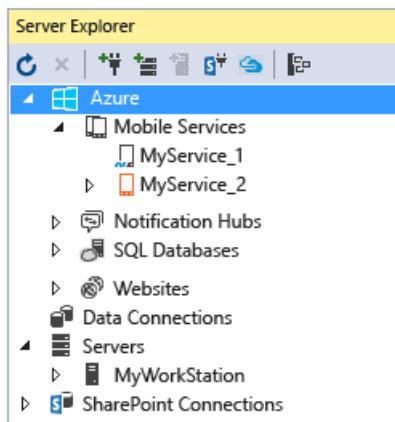
Sign in to Visual Studio

- Sign into Visual Studio with a Microsoft account or an organizational account. You should see your user name appear in the upper corner of the window, similar to this:



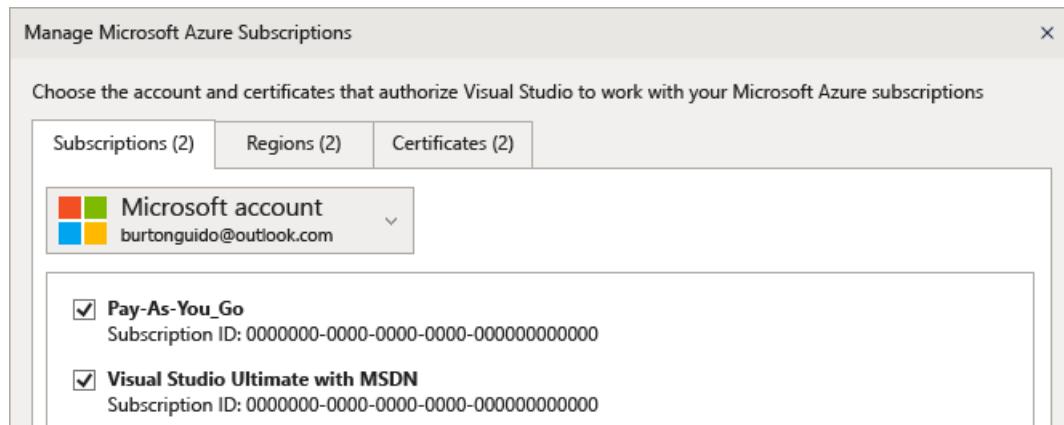
Access your Azure account in Server Explorer

Press **Ctrl + Alt + S** to open **Server Explorer**. Choose the Azure icon and when it expands you should see the resources available in the Azure account that is associated with the ID that you used to log in to Visual Studio. It should appear something like the following (except that you will see your own resources).



The first time you use Visual Studio on any specific device, the dialog will only show the subscriptions registered under the ID that you signed in to the IDE with. You can access resources for any of your other accounts directly from **Server Explorer** by right-clicking on the Azure node and choosing **Manage and Filter Subscriptions** and adding your accounts from the account picker control. You can then choose another account, if desired, by clicking the down arrow and choosing from the list of accounts. After choosing the account, you can choose which

subscriptions under that account you want to display in Server Explorer.



The next time you open Server Explorer, the resources for that subscription(s) are displayed.

Access your Azure account via Add Connected Service dialog

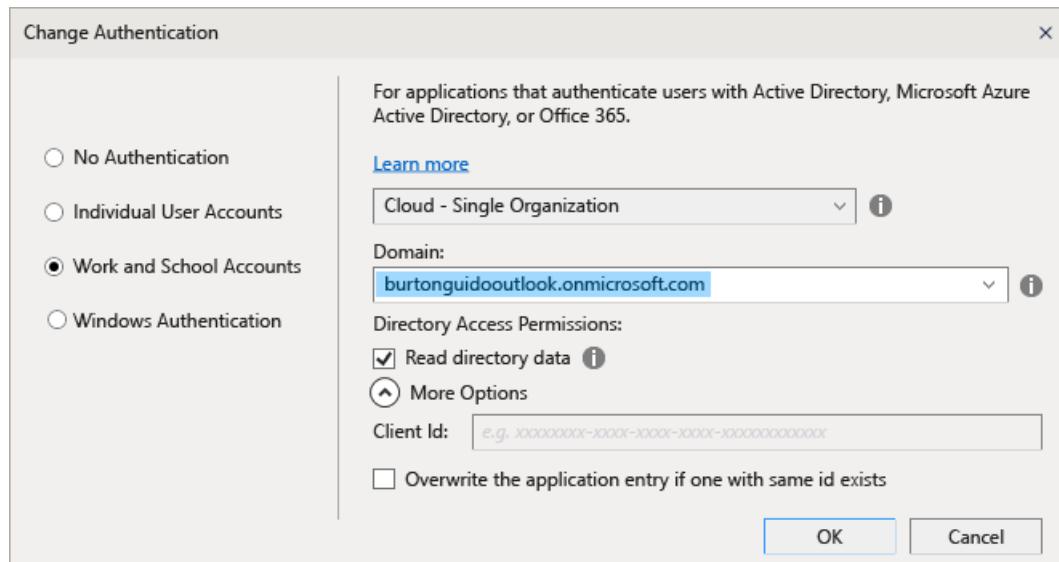
1. Create a UWP app project in C#.
2. Choose the project node in Solution Explorer and then choose **Add, Connected Service**. The **Add Connected Service** wizard appears and shows you the list of services in the Azure account that is associated with your Visual Studio login ID. Note that you do not have to sign in separately to Azure. However, you do need to sign in to the other accounts the first time you attempt to access their resources from a given computer.

WARNING

If this is the first time you are creating a UWP app in Visual Studio on a specific computer, you will be prompted to enable your device for development mode by going to **Settings | Updates and Security | For Developers** on your computer. For more information, see [Enable Your Device for Development](#).

Access Azure Active Directory in a Web project

Azure AD enables support for end-user single Sign-In in ASP.NET MVC web applications, or AD Authentication in Web API services. Domain authentication is different from individual user account authentication; users that have access to your Active Directory domain can use their existing Azure AD accounts to connect to your web applications. Office 365 apps can also use domain authentication. To see this in action, create a web application (**File, New Project, C#, Cloud, ASP.NET Web Application**). In the New ASP.NET Project dialog choose **Change Authentication**. The authentication wizard appears and enables you to choose what kind of authentication to use in your application.



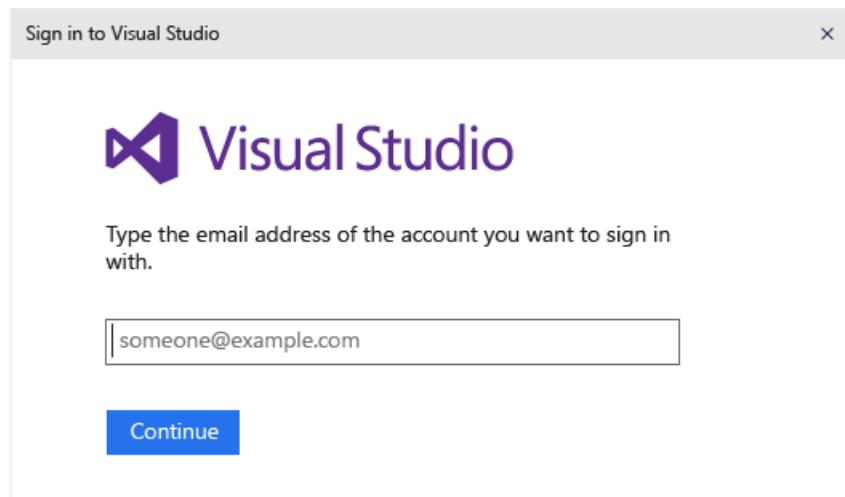
For more information about the different kinds of authentication in ASP.NET, see [Creating ASP.NET Web Projects in Visual Studio 2013](#) (the information about authentication is still relevant for current versions of Visual Studio).

Access your Visual Studio Team Services account

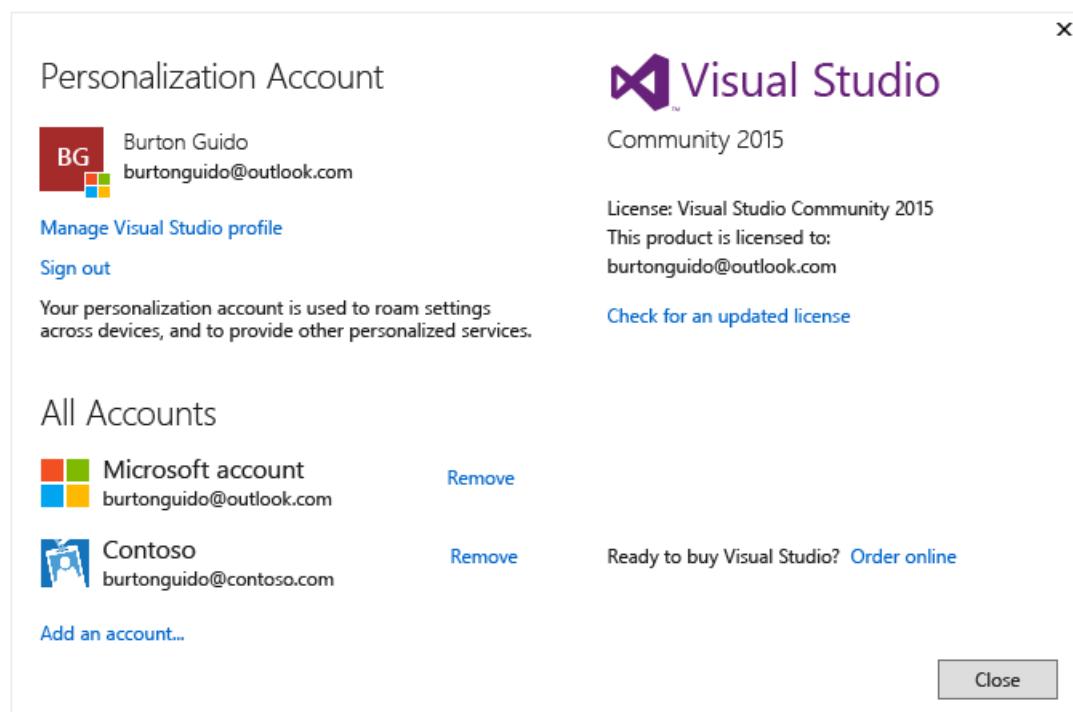
From the main menu, choose **Team, Connect to Team Foundation Server** to bring up the **Team Explorer** window. Click on **Select Team Projects**, and then in the list box under **Select a Team Foundation Server**, you should see the URL for your Visual Studio Team Services account. When you select the URL you will be logged in without having to re-enter your credentials.

Add a second user account to Visual Studio

Click on the down arrow next to your user name in the upper corner of Visual Studio. Then choose the **Account Settings** menu item. The **Account Manager** dialog appears and displays the account you signed in with. Choose the **Add an account** link in the lower corner of the dialog to add a new Microsoft account or a new work or school account.



Follow the prompts to enter the new account credentials. The following illustration shows the Account Manager after a user has added his Contoso.com work account.



Revisit the Add Connected Services Wizard and Server Explorer

Now go to **Server Explorer** again, right-click on the Azure node and choose **Manage and filter subscriptions**. Choose the new account by clicking the drop down arrow next to the current account, and then choose which subscriptions you want to display in Server Explorer. You should see all the services associated with the specified subscription. Even though you are not currently signed in to the Visual Studio IDE with the second account, you are signed in to that account's services and resources. The same is true for **Project**, **Add Connected Service** and **Team, Connect to Team Foundation Server**.

See also

[Sign in to Visual Studio](#)

How to: unlock Visual Studio

3/21/2018 • 2 min to read • [Edit Online](#)

You can evaluate Visual Studio for free up to 30 days. Signing into the IDE extends the trial period to 90 days. To continue using Visual Studio, unlock the IDE by either:

- using an online subscription
- entering a product key

To unlock Visual Studio using an online subscription

To unlock Visual Studio using an MSDN or Visual Studio Team Service subscription associated with a Microsoft account, or a work or school account:

1. Click on the "Sign in" button in the upper right corner of the IDE (or go to File > Account Settings to open the Account Settings dialog and click on the "Sign in" button).
2. Enter the credentials for either a Microsoft account or a work or school account. Visual Studio finds a Visual Studio subscription or Visual Studio Team Services subscription associated with your account.

IMPORTANT

Visual Studio automatically looks for associated online subscriptions when you connect to a Visual Studio Team Services account from the Team Explorer tool window. When you connect to a Visual Studio Team Services account, you can sign in using both Microsoft and work or school accounts. If an online subscription exists for that user account, Visual Studio will automatically unlock the IDE for you.

To unlock Visual Studio with a product key

1. Select **File, Account Settings** to open the Account Settings dialog and click on the **License with a Product Key** link.

Enter the product key in the space provided.

TIP

Pre-release versions of Visual Studio do not have product keys. You must sign in to the IDE to use pre-release versions.

Address license problem states

Update stale licenses

You may have seen the below message that your license is going stale in Visual Studio, which reads, "Your license has gone stale and must be updated."



Your license has gone stale and must be updated.
Check for an update license to continue using this product.

[Check for an updated license](#)

This message indicates that while your subscription may still be valid, the license token Visual Studio uses to keep

your subscription up to date hasn't been refreshed and has gone stale due to one of the following reasons:

- You have not used Visual Studio or have had no internet connection for an extended period of time.
- You signed out of Visual Studio.

Before the license token goes stale, Visual Studio first shows a warning message asking you to reenter your credentials.

If you do not reenter your credentials, the token starts to go stale and the Account Settings dialog tells you how many days you have left before your token will fully expire. After your token expires, you will need to reenter your credentials for this account or license with another method above before you can continue using Visual Studio.

IMPORTANT

If you are using Visual Studio for extended periods in environments with limited or no internet access, you should use a product key to unlock Visual Studio in order to avoid interruption.

Update expired licenses

If your subscription has expired completely and you no longer have access rights to Visual Studio, you must renew your subscription or add another account that has a subscription. To see more information about the license you are using, go to **File, Account Settings** and look at the license information on the right side of the dialog. If you have another subscription associated with a different account, add that account to the **All Accounts** list on the left side of the dialog box by selecting the **Add an account...** link.

See also

- [Signing in to Visual Studio](#)

Create an offline installation of Visual Studio 2017

1/22/2018 • 1 min to read • [Edit Online](#)

We designed the Visual Studio 2017 installer to work well in a wide variety of network and machine conditions.

- The new workload-based model means you'll need to download far less than with previous versions of Visual Studio: as little as 300 MB for the smallest installation;
- Compared to a generic "ISO" or zip file, we download only the packages you need for your machine. For example, we don't download 64-bit files if you don't need them;
- During the installation process, we try three different download technologies (WebClient, BITS and WinInet) to minimize interference with anti-virus and proxy software;
- The files you'll need to install Visual Studio are distributed on a global delivery network, so we can get them to you from a local server.

We recommend that you try the [Visual Studio web installer](#)—we think you'll find it a good experience.

[DOWNLOAD VISUAL STUDIO](#)

2017

If you want to install offline because your internet connection is unavailable or unreliable, see [Install Visual Studio 2017 on low bandwidth or unreliable network environments](#). You can use the command line to create a local cache of the files you need to complete an offline install. This process replaces the ISO files available for previous versions.

NOTE

If you are an enterprise administrator who wants to perform a deployment of Visual Studio 2017 to a network of client workstations that are firewalled from the internet, see our [Create a network installation of Visual Studio 2017](#) and [Install certificates required for Visual Studio offline installation](#) pages.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

Install certificates required for Visual Studio offline installation

12/22/2017 • 5 min to read • [Edit Online](#)

Visual Studio is primarily designed to be installed on an internet-connected machine, since many components are updated regularly. However, with some extra steps, it's possible to deploy Visual Studio in an environment where a working internet connection is unavailable.

The Visual Studio setup engine installs only content that is trusted. It does this by checking Authenticode signatures of the content being downloaded and verifying that all content is trusted before installing it. This keeps your environment safe from attacks where the download location is compromised. Visual Studio setup therefore requires that several standard Microsoft root and intermediate certificates are installed and up-to-date on a user's machine. If the machine has been kept up to date with Windows Update, signing certificates usually are up to date. If the machine is connected to the internet, during installation Visual Studio may refresh certificates as necessary to verify file signatures. If the machine is offline, the certificates must be refreshed another way.

How to refresh certificates when offline

There are three options for installing or updating certificates in an offline environment.

Option 1 - Manually install certificates from a layout folder

When you create a network layout, the necessary certificates are downloaded to the Certificates folder. You can then manually install the certificates by double-clicking each of the certificate files, and then clicking through the Certificate Manager wizard. If asked for a password, leave it blank.

Option 2 - Distribute trusted root certificates in an enterprise environment

For enterprises with offline machines that do not have the latest root certificates, an administrator can use the instructions on the [Configure Trusted Roots and Disallowed Certificates](#) page to update them.

Option 3 - Install certificates as part of a scripted deployment of Visual Studio

If you are scripting the deployment of Visual Studio in an offline environment to client workstations, you should follow these steps:

1. Copy the [Certificate Manager Tool](#) (certmgr.exe) to the installation share (for example, \\server\\share\\vs2017). Certmgr.exe is not included as part of Windows itself, but is available as part of the [Windows SDK](#).
2. Create a batch file with the following commands:

```

certmgr.exe -add -c certificates\manifestSignCertificates.p12 -n "Microsoft Code Signing PCA 2011" -s -
r LocalMachine CA

certmgr.exe -add -c certificates\manifestSignCertificates.p12 -n "Microsoft Root Certificate Authority"
-s -r LocalMachine root

certmgr.exe -add -c certificates\manifestCounterSignCertificates.p12 -n "Microsoft Time-Stamp PCA 2010"
-s -r LocalMachine CA

certmgr.exe -add -c certificates\manifestCounterSignCertificates.p12 -n "Microsoft Root Certificate
Authority" -s -r LocalMachine root

certmgr.exe -add -c certificates\vs_installer_opc.SignCertificates.p12 -n "Microsoft Code Signing PCA"
-s -r LocalMachine CA

certmgr.exe -add -c certificates\vs_installer_opc.SignCertificates.p12 -n "Microsoft Root Certificate
Authority" -s -r LocalMachine root

```

3. Deploy the batch file to the client. This command should be run from an elevated process.

What are the certificates files in the Certificates folder?

The three .P12 files in this folder each contain an intermediate certificate and a root certificate. Most systems that are current with Windows Update have these certificates already installed.

- **ManifestSignCertificates.p12** contains:
 - Intermediate certificate: **Microsoft Code Signing PCA 2011**
 - Not required. Improves performance in some scenarios if present.
 - Root certificate: **Microsoft Root Certificate Authority 2011**
 - Required on Windows 7 Service Pack 1 systems that do not have the latest Windows Updates installed.
- **ManifestCounterSignCertificates.p12** contains:
 - Intermediate certificate: **Microsoft Time-Stamp PCA 2010**
 - Not required. Improves performance in some scenarios if present.
 - Root certificate: **Microsoft Root Certificate Authority 2010**
 - Required for Windows 7 Service Pack 1 systems that do not have the latest Windows Updates installed.
- **Vs_installer_opc.SignCertificates.p12** contains:
 - Intermediate certificate: **Microsoft Code Signing PCA**
 - Required for all systems. Note that systems with all updates applied from Windows Update might not have this certificate.
 - Root certificate: **Microsoft Root Certificate Authority**
 - Required. This certificate ships with systems running Windows 7 or later.

Why are the certificates from the Certificates folder not installed automatically?

When a signature is verified in an online environment, Windows APIs are used to download and add the certificates to the system. Verification that the certificate is trusted and allowed via administrative settings occurs during this process. This verification process cannot occur in most offline environments. Installing the certificates manually allows enterprise administrators to ensure the certificates are trusted and meet the security policy of their organization.

Checking if certificates are already installed

One way to check on the installing system is to follow these steps:

1. Run **mmc.exe**.
 - a. Click File, and then select **Add/Remove Snap-in**.
 - b. Double-click **Certificates**, select **Computer account**, and then click **Next**.
 - c. Select **Local computer**, click **Finish**, and then click **OK**.
 - d. Expand **Certificates (Local Computer)**.
 - e. Expand **Trusted Root Certification Authorities**, and then select **Certificates**.
 - Check this list for the necessary root certificates.
 - f. Expand **Intermediate Certification Authorities**, and then select **Certificates**.
 - Check this list for the required intermediate certificates.
2. Click File and select **Add/Remove Snap-in**.
 - a. Double-click **Certificates**, select **My user account**, click **Finish**, and then click **OK**.
 - b. Expand **Certificates – Current User**.
 - c. Expand **Intermediate Certification Authorities**, and then select **Certificates**.
 - Check this list for the required intermediate certificates.

If the certificates names were not in the **Issued To** columns, they must be installed. If an intermediate certificate was only in the **Current User** Intermediate Certificate store, then it is available only to the user that is logged in. You might need to install it for other users.

Install Visual Studio

After you install the certificates, deployment of Visual Studio can proceed by using the instructions from the [Deploying from a network installation](#) section of the "Create a network installation of Visual Studio" page.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio](#)
- [Visual Studio administrator guide](#)
- [Use command-line parameters to install Visual Studio](#)
- [Visual Studio workload and component IDs](#)

Install Visual Studio 2017 on low bandwidth or unreliable network environments

3/21/2018 • 3 min to read • [Edit Online](#)

We recommend that you try the Visual Studio web installer—we think you'll find it a good experience for most situations.

[DOWNLOAD VISUAL STUDIO
2017](#)

However, if your internet connection is unavailable or unreliable, you can use the command line to create a local cache of the files you need to complete an offline install. Here's how.

NOTE

If you are an enterprise administrator who wants to perform a deployment of Visual Studio 2017 to a network of client workstations that are firewalled from the internet, see our [Create a network installation of Visual Studio 2017](#) and [Install certificates required for Visual Studio offline installation](#) pages.

Step 1 - Download the Visual Studio bootstrapper

Start by downloading the Visual Studio bootstrapper for your chosen edition of Visual Studio.

Your setup file—or to be more specific, a bootstrapper file—will match or be similar to one of the following.

| EDITION | FILE |
|----------------------------|-------------------------------------|
| Visual Studio Community | vs_community.exe |
| Visual Studio Professional | vs_professional.exe |
| Visual Studio Enterprise | vs_enterprise.exe |

Step 2 - Create a local install cache

You must have an internet connection to complete this step. To create a local layout, open a command prompt and use one of the commands from the following examples: The examples here assume that you're using the Community edition of Visual Studio; adjust the command as appropriate for your edition.

- For .NET web and .NET desktop development, run:

```
vs_community.exe --layout c:\vs2017layout --add Microsoft.VisualStudio.Workload.ManagedDesktop --add Microsoft.VisualStudio.Workload.NetWeb --add Component.GitHub.VisualStudio --includeOptional --lang en-US
```

- For .NET desktop and Office development, run:

```
vs_community.exe --layout c:\vs2017layout --add Microsoft.VisualStudio.Workload.ManagedDesktop --add Microsoft.VisualStudio.Workload.Office --includeOptional --lang en-US
```

- For C++ desktop development, run:

```
vs_community.exe --layout c:\vs2017layout --add Microsoft.VisualStudio.Workload.NativeDesktop --includeRecommended --lang en-US
```

- To create a complete local layout with all features (this will take a long time—we have *lots* of features!), run:

```
vs_community.exe --layout c:\vs2017layout --lang en-US
```

If you want to install a language other than English, change `en-US` to a locale from the list at the bottom of this page. Use this [list of the components and workloads available](#) to further customize your installation cache as necessary.

IMPORTANT

A complete Visual Studio 2017 layout requires at least 35 GB of disk space and can take some time to download. See [Use command-line parameters to install Visual Studio 2017](#) for information on how to create a layout with only the components you want to install.

Step 3 - Install Visual Studio from the local cache

TIP

When you run from a local install cache, setup uses the local versions of each of these files. But if you select components during installation that aren't in the cache, we attempt to download them from the internet.

To ensure that you only install the files you've downloaded, use the same command-line options that you used to create the layout cache. For example, if you created a layout cache with the following command:

```
vs_community.exe --layout c:\vs2017layout --add Microsoft.VisualStudio.Workload.ManagedDesktop --add  
Microsoft.VisualStudio.Workload.NetWeb --add Component.GitHub.VisualStudio --includeOptional --lang en-US
```

Use this command to run the installation:

```
c:\vs2017layout\vs_community.exe --add Microsoft.VisualStudio.Workload.ManagedDesktop --add  
Microsoft.VisualStudio.Workload.NetWeb --add Component.GitHub.VisualStudio --includeOptional
```

NOTE

If you get an error that a signature is invalid, you must install updated certificates. Open the Certificates folder in your offline cache. Double-click each of the certificate files, and then click through the Certificate Manager wizard. If asked for a password, leave it blank.

List of language locales

| LANGUAGE-LOCALE | LANGUAGE |
|-----------------|----------|
| cs-CZ | Czech |
| de-DE | German |
| en-US | English |
| es-ES | Spanish |
| fr-FR | French |
| it-IT | Italian |

| LANGUAGE-LOCALE | LANGUAGE |
|-----------------|-----------------------|
| ja-JP | Japanese |
| ko-KR | Korean |
| pl-PL | Polish |
| pt-BR | Portuguese - Brazil |
| ru-RU | Russian |
| tr-TR | Turkish |
| zh-CN | Chinese - Simplified |
| zh-TW | Chinese - Traditional |

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio](#)
- [Visual Studio administrator guide](#)
- [Use command-line parameters to install Visual Studio](#)

Troubleshooting Visual Studio 2017 installation and upgrade issues

12/22/2017 • 4 min to read • [Edit Online](#)

Symptoms

When you try to install or update Visual Studio 2017, the operation fails.

Workaround

To work around this issue, follow these steps.

Step 1 - Check whether this problem is a known issue

There are some known issues with the Visual Studio Installer that Microsoft is working on fixing. To see if there's a workaround for your problem, check the [Known Issues section of our release notes](#).

Step 2 - Check with the developer community

Search on your error message with the [Visual Studio Developer Community](#). Other members of the community may have documented a solution to your problem.

Step 3 - Delete the Visual Studio Installer directory to fix upgrade problems

The Visual Studio Installer bootstrapper is a minimal light-weight executable that installs the rest of the Visual Studio Installer. Deleting Visual Studio Installer files and then rerunning the bootstrapper might solve some update failures.

NOTE

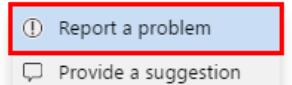
Performing the following actions reinstalls the Visual Studio Installer files and resets the installation metadata.

1. Close the Visual Studio Installer.
2. Delete the Visual Studio Installer directory. Typically, the directory is `C:\Program Files (x86)\Microsoft Visual Studio\Installer`.
3. Run the Visual Studio Installer bootstrapper. You may find the bootstrapper in your Downloads folder with a file name that follows a `vs_[Visual Studio edition]_*.*.exe` pattern. If you don't find that application, you can download the bootstrapper by going to the [Visual Studio downloads](#) page and clicking **Download** for your edition of Visual Studio. Run the executable to reset your installation metadata.
4. Try to install or update Visual Studio again. If the Installer continues to fail, go to the next step.

Step 4 - Report a problem

In some situations, such as those related to corrupted files, the problems may have to be looked at on a case-by-case basis:

1. Collect your setup logs. See [How to get the Visual Studio installation logs](#) for details.
2. Open the Visual Studio Installer, and then click **Report a problem** to open the Visual Studio Feedback tool.



Visual Studio Community 2017

Free, fully-featured IDE for students, open-source and individual developers

[License terms](#) | [Release notes](#)

(15.0.26228.4)

Welcome!

We invite you to go online to hone your skills and find additional tools to support your development workflow.

Learn

Whether you're new to development or an experienced developer, we have you covered with our tutorials, videos, and sample code.

3. Give your problem report a title, and provide relevant details. Click **Next** to go to the **Attachments** section, and then attach the generated log file (typically, the file is at `%TEMP%\vslogs.zip`).
4. Click **Next** to review your problem report, and then click **Submit**.

Step 5 - Run InstallCleanup.exe to remove installation files

As a last resort, you can [remove Visual Studio](#) to remove all installation files and product information.

1. Follow the instructions in [Remove Visual Studio](#).
2. Rerun the bootstrapper that's described in [Step 3 - Delete the Visual Studio Installer directory to fix upgrade problems](#).
3. Try to install or update Visual Studio again.

Step 6 - Contact us (optional)

If none of the other steps allow you to successfully install, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

How to troubleshoot an offline installer

Here is a table of known issues and some workarounds when installing from a local layout that might help.

| ISSUE | ITEM | SOLUTION |
|--|--------------------|---|
| Users do not have access to files. | permissions (ACLs) | Make sure that you adjust the permissions (ACLs) so that they grant Read access to other users <i>before</i> you share the offline install. |
| New workloads, components, or languages fail to install. | --layout | Make sure that you have internet access if you install from a partial layout and select workloads, components, or languages that were not downloaded previously in that partial layout. |

How to get the Visual Studio installation logs

Setup logs are needed to troubleshoot most installation issues. When you submit an issue by using [Report a Problem](#) in the Visual Studio Installer, these logs are automatically included in your report.

If you contact Microsoft Support, you may need to provide these setup logs by using the [Microsoft Visual Studio and .NET Framework Log Collection Tool](#). The log collection tool collects setup logs from all components installed by Visual Studio 2017, including .NET Framework, Windows SDK, and SQL Server. It also collects

computer information, a Windows Installer inventory, and Windows event log information for Visual Studio Installer, Windows Installer, and System Restore.

To collect the logs:

1. [Download the tool](#).
2. Open an administrative command prompt.
3. Run `Collect.exe` from the directory where you saved the tool.
4. Find the resulting `vslogs.zip` file in your `%TEMP%` directory, for example,
`C:\Users\YourName\AppData\Local\Temp\vslogs.zip`.

NOTE

The tool must be run under the same user account that the failed installation was run under. If you are running the tool from a different user account, set the `-user:<name>` option to specify the user account under which the failed installation was run. Run `Collect.exe -?` from an administrator command prompt for additional options and usage information.

More support options

If none of the other steps allow you to successfully install, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This requires a [GitHub](#) account.)

See also

- [Visual Studio Administrator Guide](#)
- [Tools for detecting and managing Visual Studio instances](#)
- [Remove Visual Studio 2017](#)

Update Visual Studio 2017 to the most recent release

3/7/2018 • 4 min to read • [Edit Online](#)

We encourage you to update to the most [recent version](#) of Visual Studio 2017 so that you always get the latest features, fixes, and improvements.

And if you'd like to try out anything before we release it, consider downloading the [preview release](#) of the next version, too.

IMPORTANT

You must log on with an account that has administrative permissions to install, update, or modify Visual Studio. For more information, see [User Permissions and Visual Studio](#).

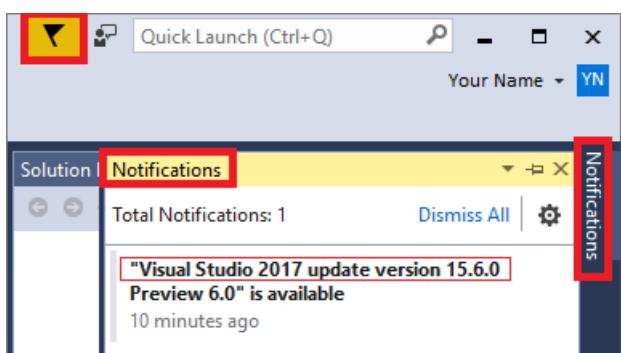
Update Visual Studio 2017 version 15.6 or later

We've streamlined the installation and update experience to make it easier to use directly from within the IDE. Here's how to update from version 15.6 and later to newer versions of Visual Studio.

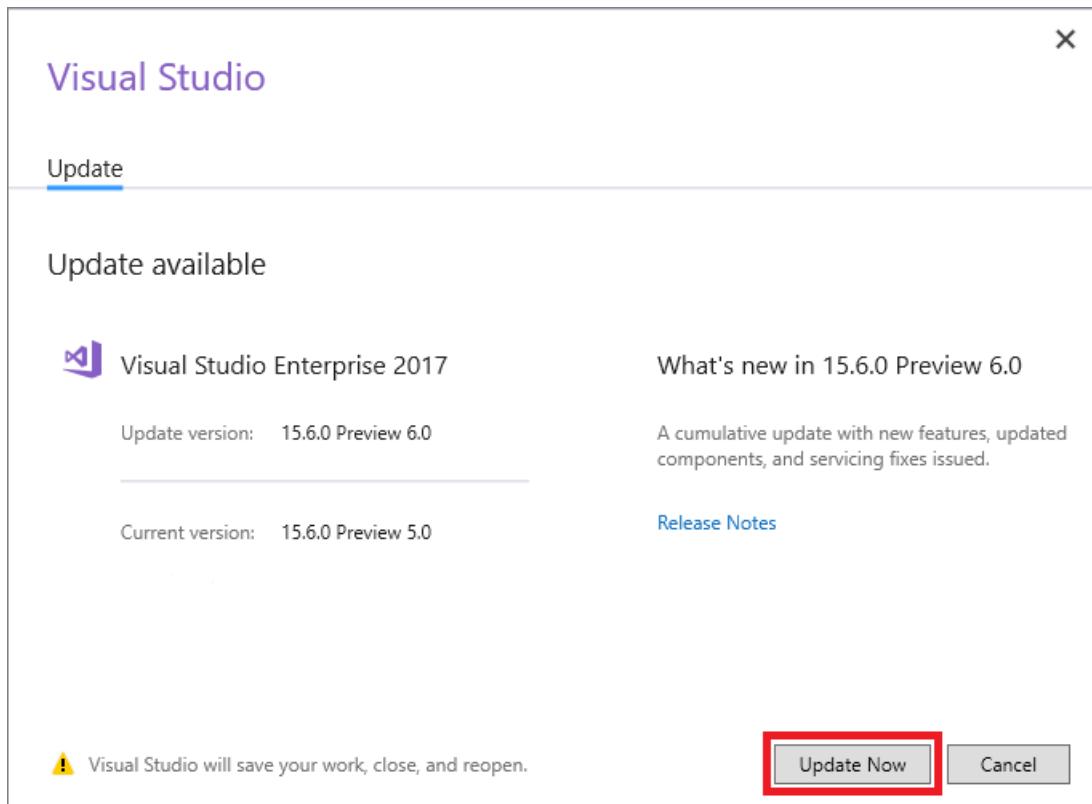
Use the Notifications hub

When there is an update, there's a corresponding notification flag in Visual Studio.

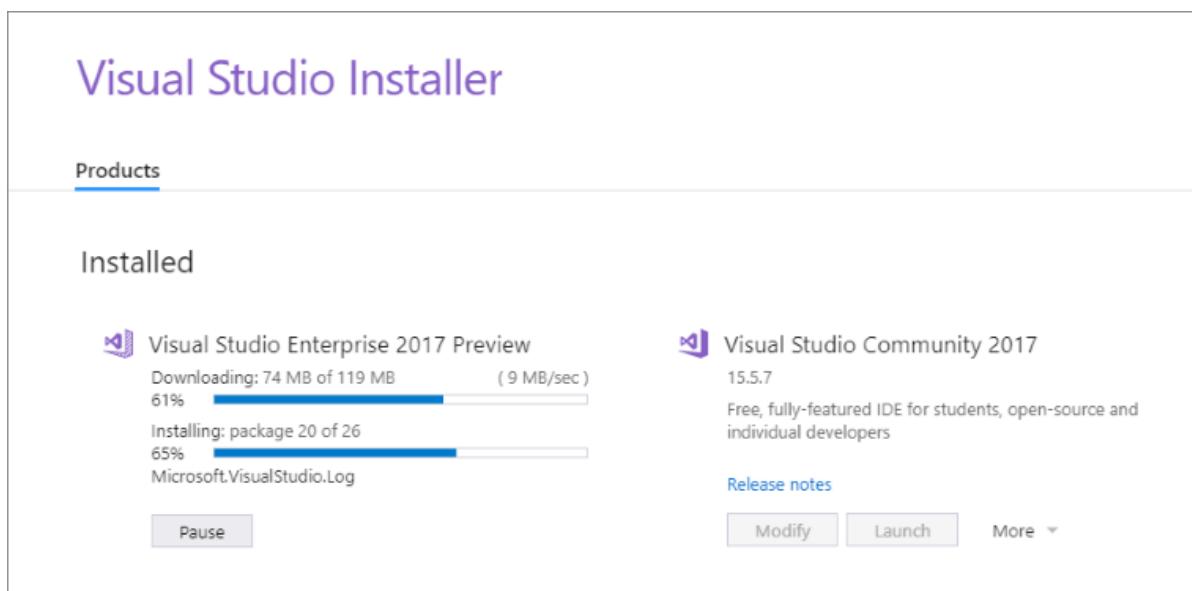
1. Save your work.
2. Choose the notification flag to open the **Notifications** hub, and then choose the update that you want to install.



3. When the **Update** dialog box opens, choose **Update Now**.



If a User Access Control dialog box opens, choose **Yes**. Next, a "Please wait" dialog might open for a moment, and then the Visual Studio Installer opens to start the update.

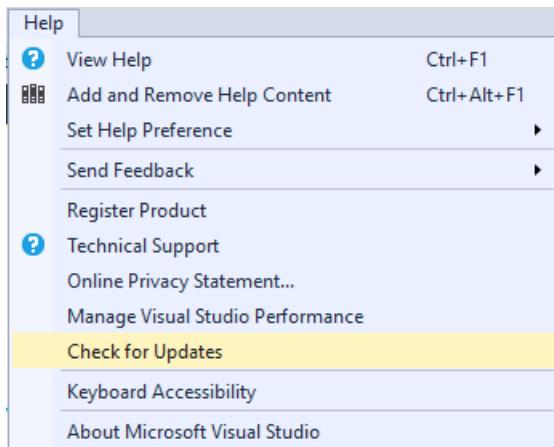


Your update continues. Then, when it's complete, Visual Studio restarts.

Use the IDE

You can check for an update and then install the update from the menu bar in Visual Studio.

1. Save your work.
2. Choose **Help > Check for Updates**.



3. When the **Update** dialog box opens, choose **Update Now**.

The update proceeds as described in the previous section, and then Visual Studio restarts after the update completes successfully.

Use the Visual Studio Installer

As in earlier versions of Visual Studio 2017, you can use the Visual Studio Installer to install an update.

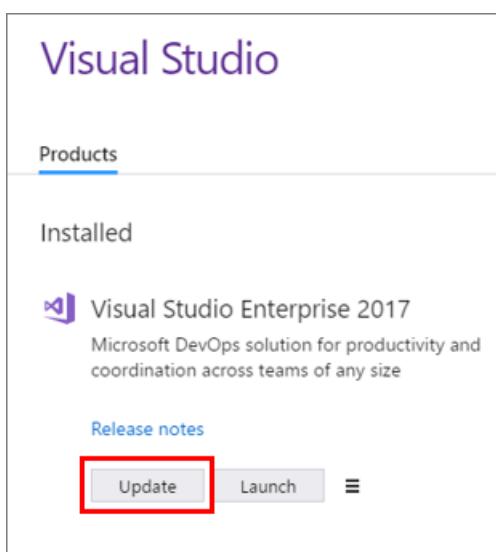
1. Save your work.
2. Open the installer. The Visual Studio Installer might require updating before you continue.

NOTE

On a computer running Windows 10, you can find the installer under the letter **V** as the **Visual Studio Installer**, or under the letter **M** as the **Microsoft Visual Studio Installer**.

3. On the **Product** page in the installer, look for the edition of Visual Studio that you have installed.
4. If an update is available, you see an **Update** button. (It might take a few seconds for the installer to determine whether an update is available.)

Choose the **Update** button to install the updates.



Update Visual Studio 2017 version 15.5 or earlier

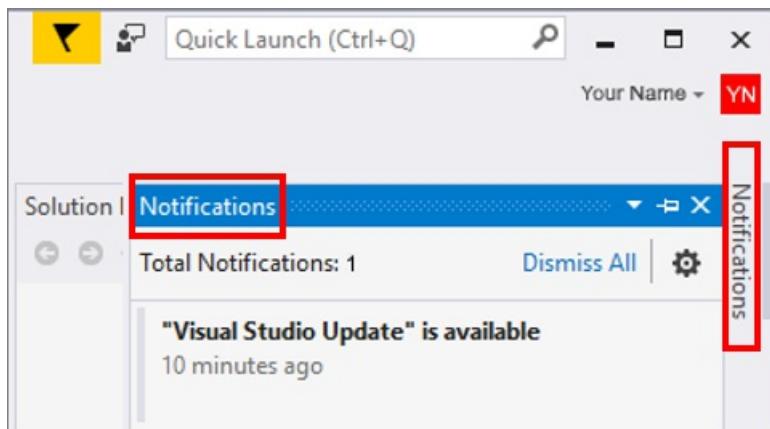
If you are using an earlier version, here's how to apply an update from Visual Studio 2017 version 15.0 through version 15.5.

Update by using the Notifications hub

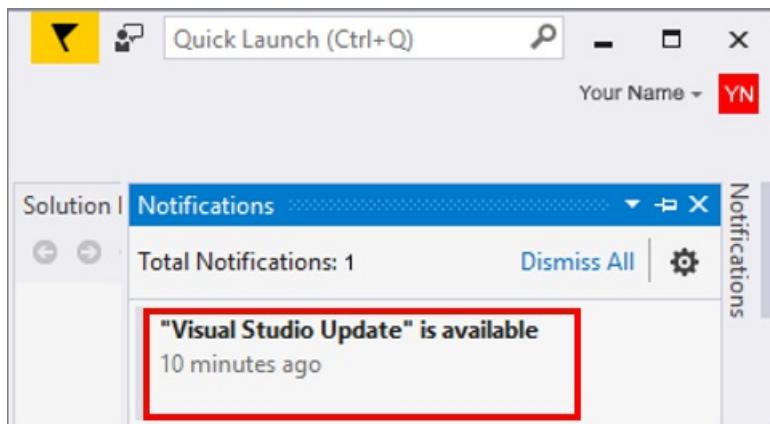
- When there are updates, there's a corresponding notification flag in Visual Studio.



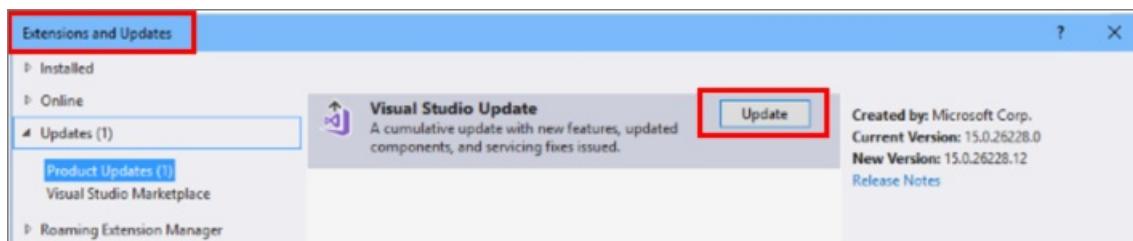
Choose the notification flag to open the **Notifications** hub.



- Choose "**Visual Studio Update**" is available, which opens the **Extensions and Updates** dialog box.



- In the **Extensions and Updates** dialog box, choose the **Update** button.



More about Visual Studio notifications

Visual Studio notifies you when an update is available for Visual Studio itself or for any components, and also when certain events occur in the Visual Studio environment.

- When the notification flag is yellow, there is a Visual Studio product update available for you to install.
- When the notification flag is red, there is a problem with your license.
- When the notification flag is black, there are optional or informational messages to review.

Choose the notifications flag to open the **Notifications** hub and then choose the notifications that you want to act on. Or, choose to ignore or dismiss a notification.



If you choose to ignore a notification, Visual Studio stops showing it. If you want to reset the list of ignored notifications, choose the **Settings** button in the Notifications hub.



Update by using the Visual Studio Installer

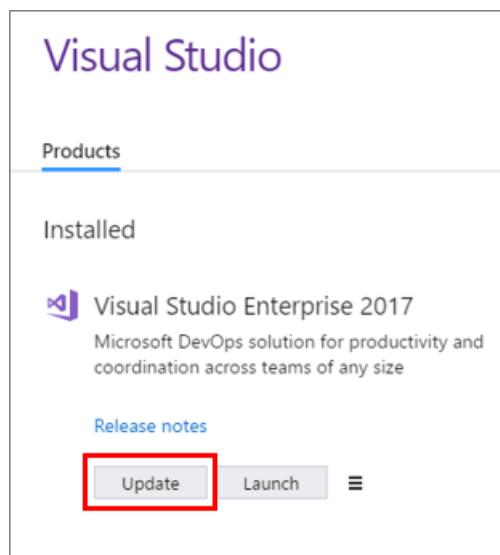
1. Open the installer. You might need to update the installer before continuing. If this is the case, you are prompted to do so.

NOTE

On a computer running Windows 10, you can find the installer under the letter **V** as the **Visual Studio Installer**, or under the letter **M** as the **Microsoft Visual Studio Installer**.

2. On the **Product** page in the installer, look for the edition of Visual Studio that you have installed.
3. If an update is available, you see an **Update** button. (It might take a few seconds for the installer to determine whether an update is available.)

Choose the **Update** button to install the updates.



Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio 2017](#)
- [Modify Visual Studio 2017](#)
- [Uninstall Visual Studio 2017](#)
- [Visual Studio Administrators Guide](#)

Modify Visual Studio 2017 by adding or removing workloads and components

12/22/2017 • 2 min to read • [Edit Online](#)

Not only have we made it easier for you to personalize Visual Studio to match the tasks you want to accomplish, we've also made it easier to customize Visual Studio, too. No more looking in Control Panel to do so; instead, start the new Visual Studio Installer and make the changes you want.

Here's how.

Modify workloads

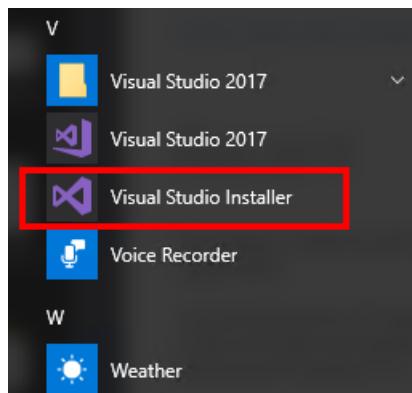
Workloads contain the features you need for the programming language or platform that you are using. Use workloads to modify Visual Studio so that it supports the work you want to do, when you want to do it.

IMPORTANT

To install, update, or modify Visual Studio, you must log on with an account that has administrative permissions. For more information, see [User Permissions and Visual Studio](#).

1. Find the Visual Studio Installer on your computer.

For example, on a computer running Windows 10 Anniversary Update, select **Start**, and then scroll to the letter **V**, where it's listed as **Visual Studio Installer**.



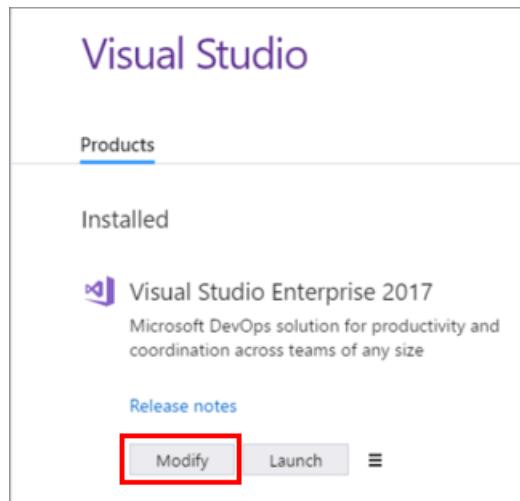
NOTE

On some computers, the Visual Studio Installer might be listed under the letter "M" as the **Microsoft Visual Studio Installer**.

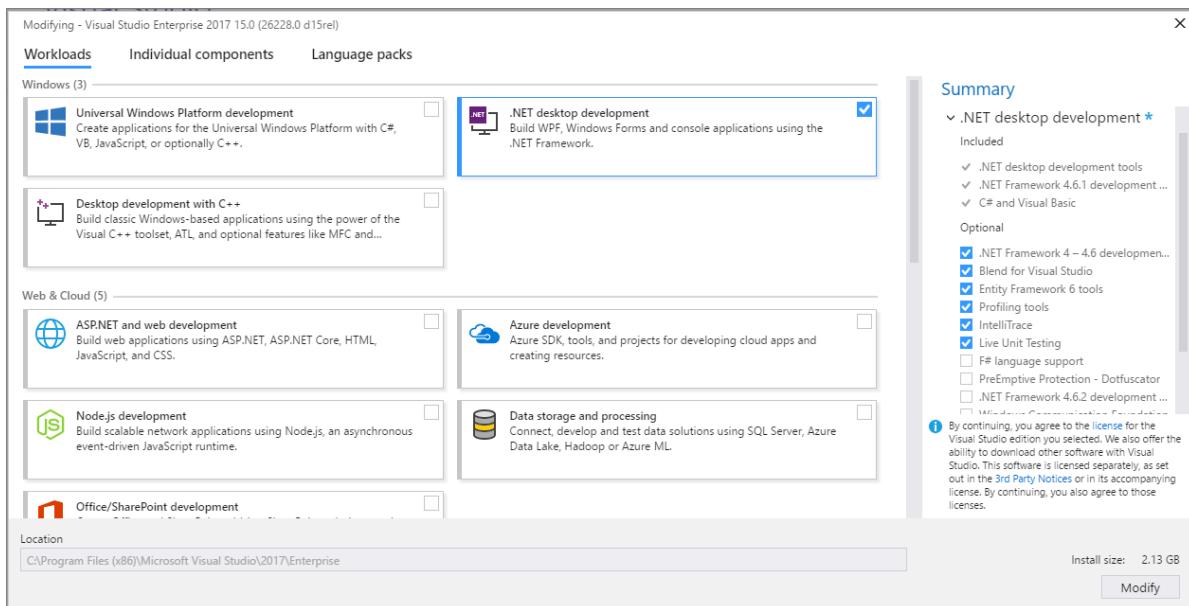
Alternatively, you can find the Visual Studio Installer in the following location:

```
C:\Program Files (x86)\Microsoft Visual Studio\Installer\vs_installer.exe
```

2. Click or tap to start the installer, and then select **Modify**.



3. From the **Workloads** screen, select or deselect the workloads that you want to install or uninstall.



4. Click or tap **Modify** again.

5. After the new workloads and components are installed, click **Launch**.

Modify individual components

If you don't want to use the handy Workloads feature to customize your Visual Studio installation, choose the **Individual Components** option from the Visual Studio Installer, select what you want, and then follow the prompts.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the](#)

[Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio 2017](#)
- [Update Visual Studio 2017](#)
- [Uninstall Visual Studio 2017](#)

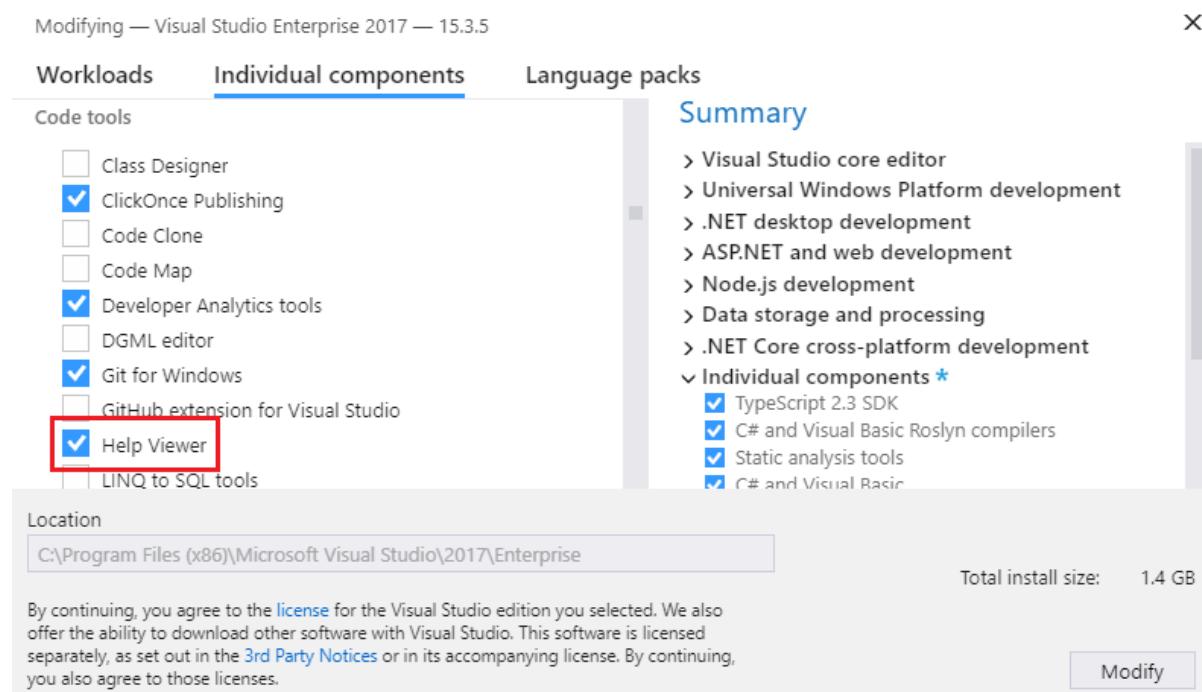
Microsoft Help Viewer installation

1/26/2018 • 1 min to read • [Edit Online](#)

Several products can display Help content in Microsoft Help Viewer, including Visual Studio and SQL Server.

Help Viewer is an optional installation component of Visual Studio. To install it through Visual Studio Installer, follow these steps:

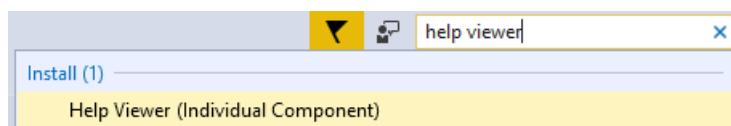
1. Open **Visual Studio Installer** from the Start menu or, if you have Visual Studio open, you can choose **Tools, Get Tools and Features...** to open Visual Studio Installer.
2. Choose the **Individual Components** tab, then select **Help Viewer** under the **Code tools** section.



3. Choose the **Modify** button to start the installation of Microsoft Help Viewer.

Another way to easily install Microsoft Help Viewer is via the **Quick Launch** box:

1. Type or enter **help viewer** in the **Quick Launch** box on the Visual Studio title bar.



2. Choose the Install result called **Help Viewer (Individual Component)**.
3. In the dialog box that opens, choose the **Install** button.

The workloads and components that you selected will be installed. We might need to restart Visual Studio or your computer during the installation.

Individual components

Help Viewer

Total install size: 1.4 GB

[Customize](#)

By continuing, you agree to the [license](#) for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the [3rd Party Notices](#) or in its accompanying license. By continuing, you also agree to those licenses.

[Install](#)

[Cancel](#)



[Watch a video](#) on how to install the Microsoft Help Viewer by using Visual Studio Installer.

See also

[Microsoft Help Viewer](#)

[Help viewer and offline content for SQL Server](#)

Repair Visual Studio 2017

3/21/2018 • 1 min to read • [Edit Online](#)

Sometimes your Visual Studio installation becomes damaged or corrupted. A repair can fix this.

1. Find the Visual Studio Installer on your computer.

For example, on a computer running Windows 10 Anniversary Update, select **Start**, and then scroll to the letter **V**, where it's listed as **Visual Studio Installer**.

NOTE

On some computers, the Visual Studio Installer might be listed under the letter "**M**" as the **Microsoft Visual Studio Installer**.

Alternatively, you can find the Visual Studio Installer in the following location:

```
C:\Program Files (x86)\Microsoft Visual Studio\Installer\vs_installer.exe
```

2. Click or tap to start the installer, then select **More**, and then select **Repair**.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio 2017](#)
- [Update Visual Studio 2017](#)
- [Uninstall Visual Studio 2017](#)
- [Troubleshooting Visual Studio 2017 installation and upgrade issues](#)

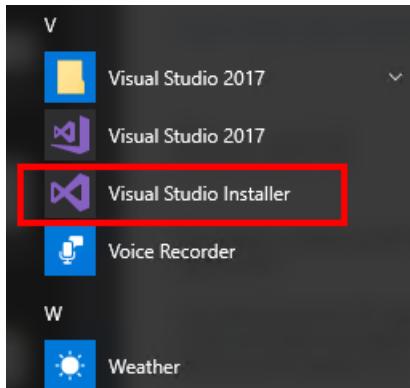
Uninstall Visual Studio

12/22/2017 • 1 min to read • [Edit Online](#)

This page walks you through uninstalling Visual Studio, our integrated suite of productivity tools for developers.

1. Find the Visual Studio Installer on your computer.

For example, on a computer running Windows 10 Anniversary Update, select **Start** and scroll to the letter **V**, where it is listed as **Visual Studio Installer**.



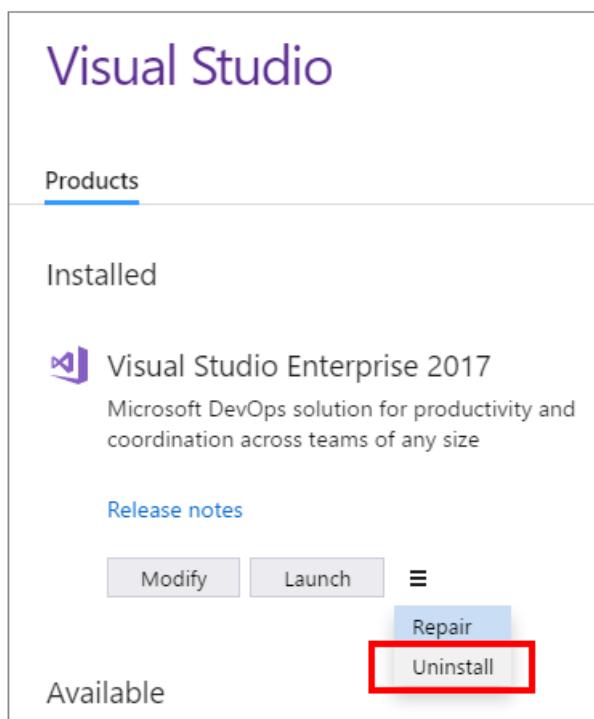
NOTE

On some computers, the Visual Studio Installer might be listed under the letter "**M**" as the **Microsoft Visual Studio Installer**.

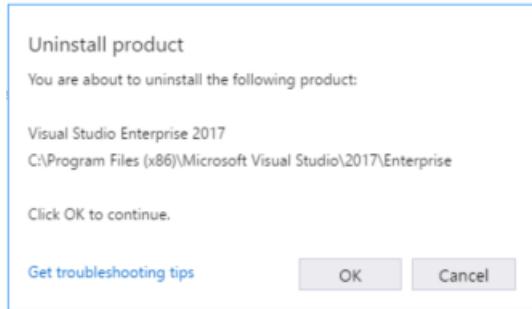
Alternatively, you can find the Visual Studio Installer in the following location:

```
C:\Program Files (x86)\Microsoft Visual Studio\Installer\vs_installer.exe
```

2. Click to start the installer, click the **≡** icon, and then click **Uninstall**.



3. Click **OK** to confirm your choice.



If you change your mind later and want to reinstall Visual Studio 2017, start the Visual Studio Installer again, and then select **Install** from the selection screen.

Uninstall Visual Studio Installer

To completely remove all installations of Visual Studio 2017 as well as the Visual Studio Installer from your machine, uninstall it from Apps & Features.

1. Open Apps & Features. For example, in Windows 10, select **Start**, and in the search bar, type **Apps and Features**.
2. Find **Microsoft Visual Studio 2017**.
3. Click **Uninstall**.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio](#)
- [Modify Visual Studio 2017](#)
- [Update Visual Studio](#)
- [Remove Visual Studio](#)

Visual Studio 2017 administrator guide

12/22/2017 • 3 min to read • [Edit Online](#)

In enterprise environments, it's common for system administrators to deploy installations to end-users from a network share or by using systems management software. We've designed the Visual Studio setup engine to support enterprise deployment, allowing system administrators the ability to create a network install location, to pre-configure installation defaults, to deploy product keys during the installation process, and to manage product updates after a successful rollout. This administrator guide provides scenario-based guidance for enterprise deployment in networked environments.

Deploying Visual Studio 2017 in an enterprise environment

You can deploy Visual Studio 2017 to client workstations as long as each target computer meets the [minimum installation requirements](#). Whether you're deploying through software like System Center or through a batch file, you'll typically want to go through the following steps:

1. [Create a network share that contains the Visual Studio product files](#) to a network location.
2. [Select the workloads and components](#) you want to install.
3. [Create a response file](#) that contains default installation options. Or alternatively, [build an installation script](#) that uses command-line parameters to control the installation.
4. Optionally, [apply a volume license product key](#) as part of the installation script so that users don't need to activate the software separately.
5. Update the network layout to [control when product updates are delivered to your end-users](#).
6. Optionally, set registry keys to [control what is cached on client workstations](#).
7. Use your deployment technology of choice to execute the script generated in the previous steps on your target developer workstations.
8. [Refresh your network location with the latest updates](#) to Visual Studio by running the command you used in step 1 on a regular basis to add updated components.

IMPORTANT

Note that installations from a network share will "remember" the source location they came from. This means that a repair of a client machine might need to return to the network share that the client originally installed from. Choose your network location carefully so that it aligns to the lifetime that you expect to have Visual Studio 2017 clients running in your organization.

Visual Studio tools

We have several tools available to help you [detect and manage installed Visual Studio instances](#) on client machines.

TIP

In addition to the documentation in the administrator guide, a good source of information on Visual Studio 2017 setup is [Heath Stewart's blog](#).

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio 2017](#)
- [Use command-line parameters to install Visual Studio 2017](#)
 - [Command-line parameter examples](#)
 - [Workload and Component ID reference](#)
- [Create a networked-based installation of Visual Studio](#)
 - [Install certificates required for Visual Studio offline installation](#)
- [Automate Visual Studio with a response file](#)
- [Automatically apply product keys when deploying Visual Studio](#)
- [Set defaults for enterprise deployments of Visual Studio](#)
- [Disable or move the package cache](#)
- [Update a networked-based installation of Visual Studio](#)
- [Control updates to Visual Studio deployments](#)
- [Tools for detecting and managing Visual Studio instances](#)

Use command-line parameters to install Visual Studio 2017

1/22/2018 • 9 min to read • [Edit Online](#)

When you install Visual Studio 2017 from a command prompt, you can use a variety of command-line parameters to control or customize the installation. From the command line, you can perform the following actions:

- Start the install with certain options preselected.
- Automate the installation process.
- Create a cache (layout) of the installation files for later use.

The command-line options are used in conjunction with the setup bootstrapper, which is the small (approximately 1MB) file that initiates the download process. The bootstrapper is the first executable that is launched when you download from the Visual Studio site. Use the following links to get a direct link to the latest release bootstrapper for the product edition that you're installing:

- [Visual Studio 2017 Enterprise](#)
- [Visual Studio 2017 Professional](#)
- [Visual Studio 2017 Community](#)

List of command-line parameters

Visual Studio command-line parameters are case-insensitive.

Syntax: `vs_enterprise.exe [command] <options>...`

(Replace `vs_enterprise.exe` as appropriate for the product edition you're installing.)

TIP

For more examples of how to use the command line to install Visual Studio 2017, see the [Command-line parameter examples](#) page.)

| COMMAND | DESCRIPTION |
|------------------------|----------------------------------|
| (blank) | Installs the product. |
| <code>modify</code> | Modifies an installed product. |
| <code>update</code> | Updates an installed product. |
| <code>repair</code> | Repairs an installed product. |
| <code>uninstall</code> | Uninstalls an installed product. |

| INSTALL OPTION | DESCRIPTION |
|---|--|
| <code>--installPath <dir></code> | The installation directory for the instance to act upon. For the install command, this is Optional and is where the instance will be installed. For other commands, this is Required and is where the previously installed instance was installed. |
| <code>--addProductLang <language-locale></code> | Optional: During an install or modify operation, this determines the UI language packs that are installed to the product. It can appear multiple times on the command line to add multiple language packs. If not present, the installation uses the machine locale. For more information, see the List of language locales section on this page. |
| <code>--removeProductLang <language-locale></code> | Optional: During an install or modify operation, this determines the UI language packs that are to be removed from the product. It can appear multiple times on the command line to add multiple language packs. For more information, see the List of language locales section on this page. |
| <code>--add <one or more workload or component IDs></code> | Optional: One or more workload or component IDs to add. The required components of the artifact are installed, but not the recommended or optional components. You can control additional components globally using <code>--includeRecommended</code> and/or <code>--includeOptional</code> . For finer-grained control, you can append <code>;includeRecommended</code> or <code>;includeOptional</code> to the ID (for example, <code>--add Workload1;includeRecommended</code> or <code>--add Workload2;includeRecommended;includeOptional</code>). For more information, see our Workload and component IDs page. You can repeat this option as necessary. |
| <code>--remove <one or more workload or component IDs></code> | Optional: One or more workload or component IDs to remove. For more information, see our Workload and component IDs page. You can repeat this option as necessary. |
| <code>--in <path></code> | Optional: The URI or path to a response file. |
| <code>--all</code> | Optional: Whether to install all workloads and components for a product. |
| <code>--allWorkloads</code> | Optional: Installs all workloads and components, no recommended or optional components. |
| <code>--includeRecommended</code> | Optional: Includes the recommended components for any workloads that are installed, but not the optional components. The workloads are specified either with <code>--allWorkloads</code> or <code>--add</code> . |
| <code>--includeOptional</code> | Optional: Includes the optional components for any workloads that are installed, but not the recommended components. The workloads are specified either with <code>--allWorkloads</code> or <code>--add</code> . |

| INSTALL OPTION | DESCRIPTION |
|---------------------|--|
| --quiet, -q | Optional: Do not display any user interface while performing the installation. |
| --passive, -p | Optional: Display the user interface, but do not request any interaction from the user. |
| --norestart | Optional: If present, commands with --passive or --quiet will not automatically restart the machine (if required). This is ignored if neither --passive nor --quiet are specified. |
| --nickname <name> | Optional: This defines the nickname to assign to an installed product. The nickname cannot be longer than 10 characters. |
| --productKey | Optional: This defines the product key to use for an installed product. It is composed of 25 alphanumeric characters either in the format xxxxx-xxxxx-xxxxx-xxxxx-xxxxx orxxxxxxxxxxxxxxxxxxxxx . |
| --help, --?, -h, -? | Display an offline version of this page. |

Note: When specifying multiple workloads and components, you must repeat the --add or --remove command-line switch for each item.

| LAYOUT OPTIONS | DESCRIPTION |
|---|---|
| --layout <dir> | Specifies a directory to create an offline install cache. For more information, see Create a network-based installation of Visual Studio . |
| --lang <one or more language-locales> | Optional: Used with --layout to prepare an offline install cache with resource packages with the specified language(s). For more information, see the List of language locales section on this page. |
| --add <one or more workload or component IDs> | Optional: One or more workload or component IDs to add. The required components of the artifact are installed, but not the recommended or optional components. You can control additional components globally using --includeRecommended and/or --includeOptional . For finer-grained control, you can append ;includeRecommended or ;includeOptional to the ID (for example, --add Workload1;includeRecommended or --add Workload2;includeOptional). For more information, see our Workload and component IDs page. Note: If --add is used, only the specified workloads and components and their dependencies are downloaded. If --add is not specified, all workloads and components are downloaded to the layout. |

| LAYOUT OPTIONS | DESCRIPTION |
|---|--|
| --includeRecommended | Optional: Includes the recommended components for any workloads that are installed, but not the optional components. The workloads are specified either with --allWorkloads or --add. |
| --includeOptional | Optional: Includes the recommended <i>and</i> optional components for any workloads being included in the layout. The workloads are specified with --add. |
| --keepLayoutVersion | New in 15.3, optional: Apply changes to the layout without updating the version of the layout. |
| --verify | New in 15.3, optional: Verify the contents of a layout. Any corrupt or missing files are listed. |
| --fix | New in 15.3, optional: Verify the contents of a layout. If any files are found to be corrupt or missing, they are re-downloaded. Internet access is required to fix a layout. |
| --clean <one or more paths to catalogs> | New in 15.3, optional: Removes old versions of components from a layout that has been updated to a newer version. |
| ADVANCED INSTALL OPTIONS | DESCRIPTION |
| --channelId <id> | Optional: The ID of the channel for the instance to be installed. This is required for the install command, ignored for other commands if --installPath is specified. |
| --channelUri <uri> | Optional: The URI of the channel manifest. If updates are not desired, --channelUri can point to a non-existent file. (for example, --channelUri C:\doesntExist.chman) This can be used for the install command; it is ignored for other commands. |
| --installChannelUri <uri> | Optional: The URI of the channel manifest to use for the installation. The URI specified by --channelUri (which must be specified when --installChannelUri is specified) is used to detect updates. This can be used for the install command; it is ignored for other commands. |
| --installCatalogUri <uri> | Optional: The URI of the catalog manifest to use for the installation. If specified, the channel manager attempts to download the catalog manifest from this URI before using the URI in the install channel manifest. This parameter is used to support offline install, where the layout cache will be created with the product catalog already downloaded. This can be used for the install command; it is ignored for other commands. |
| --productId <id> | Optional The ID of the product for the instance that will be installed. This is prepopulated in normal installation conditions. |

| ADVANCED INSTALL OPTIONS | DESCRIPTION |
|----------------------------|---|
| --wait | Optional: The process will wait until the install is completed before returning an exit code. This is useful when automating installations where one needs to wait for the install to finish to handle the return code from that install. |
| --locale <language-locale> | Optional: Change the display language of the user interface for the installer itself. Setting will be persisted. For more information, see the List of language locales section on this page. |
| --cache | New in 15.2, optional: If present, packages will be kept after being installed for subsequent repairs. This overrides the global policy setting to be used for subsequent installs, repairs, or modifications. The default policy is to cache packages. This is ignored for the uninstall command. Read how to disable or move the package cache for more information. |
| --nocache | New in 15.2, optional: If present, packages will be deleted after being installed or repaired. They will be downloaded again only if needed and deleted again after use. This overrides the global policy setting to be used for subsequent installs, repairs, or modifications. The default policy is to cache packages. This is ignored for the uninstall command. Read how to disable or move the package cache for more information. |
| --noUpdateInstaller | New in 15.2, optional: If present, prevents the installer from updating itself when quiet is specified. The installer will fail the command and return a non-zero exit code if noUpdateInstaller is specified with quiet when an installer update is required. |
| --noWeb | New in 15.3, optional: Setup now downloads any content that it is installing from the Internet. All content that is being installed must be available in an offline layout. If the layout is missing content, setup fails. For more information, see Deploying from a network installation . |

List of workload IDs and component IDs

For a list of workload and component IDs sorted by Visual Studio product, see the [Visual Studio 2017 Workload and Component IDs](#) page.

List of language locales

| LANGUAGE-LOCALE | LANGUAGE |
|-----------------|----------|
| cs-CZ | Czech |
| de-DE | German |
| en-US | English |
| es-ES | Spanish |

| LANGUAGE-LOCALE | LANGUAGE |
|-----------------|-----------------------|
| fr-FR | French |
| it-IT | Italian |
| ja-JP | Japanese |
| ko-KR | Korean |
| pl-PL | Polish |
| pt-BR | Portuguese - Brazil |
| ru-RU | Russian |
| tr-TR | Turkish |
| zh-CN | Chinese - Simplified |
| zh-TW | Chinese - Traditional |

Error codes

Depending on the result of the operation, the `%ERRORLEVEL%` environment variable will be set to one of the following values:

| VALUE | RESULT |
|-------|---|
| 0 | Operation completed successfully |
| 1602 | Operation was canceled |
| 3010 | Operation completed successfully, but install requires reboot before it can be used |
| 5004 | Operation was canceled |
| 5007 | Operation was blocked - the computer does not meet the requirements |
| Other | Failure condition occurred - check the logs for more information |

Each operation generates several log files in the `%TEMP%` directory that indicate the progress of the installation. Sort the folder by date and look for files that begin with `dd_bootstrapper`, `dd_client`, and `dd_setup` for the bootstrapper, the installer app, and the setup engine, respectively.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio](#)

[2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Command-line parameter examples for Visual Studio 2017 installation](#)
- [Create an offline installation of Visual Studio 2017](#)
- [Automate Visual Studio installation with a response file](#)

Command-line parameter examples for Visual Studio 2017 installation

3/21/2018 • 3 min to read • [Edit Online](#)

To illustrate how to [use command-line parameters to install Visual Studio](#), here are several examples that you can customize to match your needs.

In each example, `vs_enterprise.exe`, `vs_professional.exe` and `vs_community.exe` represent the respective edition of the Visual Studio bootstrapper, which is the small (approximately 1MB) file that initiates the download process. If you are using a different edition, substitute the appropriate bootstrapper name.

NOTE

All commands require administrative elevation, and a User Account Control prompt will be displayed if the process is not started from an elevated prompt.

NOTE

You can use the `^` character at the end of a command line to concatenate multiple lines into a single command. Alternatively, you can simply place these lines together onto a single row. In PowerShell, the equivalent is the backtick (```) character.

- Install a minimal instance of Visual Studio, with no interactive prompts but progress displayed:

```
vs_enterprise.exe --installPath C:\minVS ^
--add Microsoft.VisualStudio.Workload.CoreEditor ^
--passive --norestart
```

- Update a Visual Studio instance by using the command line, with no interactive prompts but progress displayed:

```
vs_enterprise.exe --update --quiet --wait
vs_enterprise.exe update --wait --passive --norestart --installPath "C:\installPathVS"
```

NOTE

Both commands are required. The first command updates the Visual Studio Installer. The second command updates the Visual Studio instance. To avoid a User Account Control dialog, run the command prompt as an Administrator.

- Install a desktop instance of Visual Studio silently, with the French language pack, returning only when the product is installed.

```
vs_enterprise.exe --installPath C:\desktopVS ^
--addProductLang fr-FR ^
--add Microsoft.VisualStudio.Workload.ManagedDesktop ^
--includeRecommended --quiet --wait
```

NOTE

The `--wait` parameter is designed for use in a batch file. In a batch file, execution of the next command will not continue until the installation has completed. The `%ERRORLEVEL%` environment variable will contain the return value of the command, as documented in the [Use command-line parameters to install Visual Studio](#) page.

- Download the Visual Studio core editor (the most minimal Visual Studio configuration). Only include the English language pack:

```
vs_community.exe --layout C:\VS2017  
--lang en-US ^  
--add Microsoft.VisualStudio.Workload.CoreEditor
```

- Download the .NET desktop and .NET web workloads along with all recommended components and the GitHub extension. Only include the English language pack:

```
vs_community.exe --layout C:\VS2017 ^  
--lang en-US ^  
--add Microsoft.VisualStudio.Workload.NetWeb ^  
--add Microsoft.VisualStudio.Workload.ManagedDesktop ^  
--add Component.GitHub.VisualStudio ^  
--includeRecommended
```

- Start an interactive installation of all workloads and components that are available in the Visual Studio 2017 Enterprise edition:

```
vs_enterprise.exe --all --includeRecommended --includeOptional
```

- Install a second, named instance of Visual Studio 2017 Professional on a machine with Visual Studio 2017 Community edition already installed, with support for Node.js development:

```
vs_professional.exe --installPath C:\VSforNode ^  
--add Microsoft.VisualStudio.Workload.Node --includeRecommended --nickname VSforNode
```

- Remove the Profiling Tools component from the default installed Visual Studio instance:

```
vs_enterprise.exe modify ^  
--installPath "C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise" ^  
--remove Microsoft.VisualStudio.Component.DiagnosticTools ^  
--passive
```

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).

- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Visual Studio Administrator Guide](#)
- [Use command-line parameters to install Visual Studio](#)
- [Create an offline installation of Visual Studio 2017](#)

Create a network installation of Visual Studio 2017

12/22/2017 • 7 min to read • [Edit Online](#)

Commonly, an enterprise administrator creates a network install point for deployment to client workstations. We've designed Visual Studio 2017 to enable you to cache the files for the initial installation along with all product updates to a single folder. (This process is also referred to as *creating a layout*.) We've done this so that client workstations can use the same network location to manage their installation even if they haven't yet updated to the latest servicing update.

NOTE

If you have multiple editions of Visual Studio in use within your enterprise (for example, both Visual Studio Professional and Visual Studio Enterprise), you must create a separate network install share for each edition.

Download the Visual Studio bootstrapper

Download the edition of Visual Studio you want. Make sure to click **Save**, and then click **Open folder**.

Your setup executable—or to be more specific, a bootstrapper file—should match one of the following.

| EDITION | DOWNLOAD |
|----------------------------|-------------------------------------|
| Visual Studio Enterprise | vs_enterprise.exe |
| Visual Studio Professional | vs_professional.exe |
| Visual Studio Community | vs_community.exe |

Other supported bootstrappers include [vs_buildtools.exe](#), [vs_feedbackclient.exe](#), [vs_teamexplorer.exe](#), [vs_testagent.exe](#), [vs_testcontroller.exe](#), and [vs_testprofessional.exe](#).

Create an offline installation folder

You must have an internet connection to complete this step. To create an offline installation with all languages and all features, use one of the commands from the following examples.

IMPORTANT

A complete Visual Studio 2017 layout requires at least 35 GB of disk space and can take some time to download. See the [Customizing the network layout](#) section for details on how to create a layout with only the components you want to install.

TIP

Make sure that you run the command from your Download directory. Typically, that's `C:\Users\<username>\Downloads` on a computer running Windows 10.

- For Visual Studio Enterprise, run:

```
vs_enterprise.exe --layout c:\vs2017offline
```

- For Visual Studio Professional, run:

```
vs_professional.exe --layout c:\vs2017offline
```

- For Visual Studio Community, run:

```
vs_community.exe --layout c:\vs2017offline
```

Modify the response.json file

You can modify the response.json to set default values that are used when setup is run. For example, you can configure the `response.json` file to select a specific set of workloads selected automatically. See [Automate Visual Studio installation with a response file](#) for details.

Copy the layout to a network share

Host the layout on a network share so it can be run from other machines.

- Example:

```
xcopy /e c:\vs2017offline \\server\products\VS2017
```

Customizing the network layout

There are several options you can use to customize your network layout. You can create a partial layout that only contains a specific set of [language locales, workloads, components, and their recommended or optional dependencies](#). This might be useful if you know that you are going to deploy only a subset of workloads to client workstations. Typical command-line parameters for customizing the layout include:

- `--add` to specify [workload or component IDs](#). If `--add` is used, only those workloads and components specified with `--add` are downloaded. If `--add` is not used, all workload and components are downloaded.
- `--includeRecommended` to include all the recommended components for the specified workload IDs
- `--includeOptional` to include all the recommended and optional components for the specified workload IDs.
- `--lang` to specify [language locales](#).

Here are a few examples of how to create a custom partial layout.

- To download all workloads and components for only one language, run:

```
vs_enterprise.exe --layout C:\vs2017offline --lang en-US
```

- To download all workloads and components for multiple languages, run:

```
vs_enterprise.exe --layout C:\vs2017offline --lang en-US de-DE ja-JP
```

- To download one workload for all languages, run

```
vs_enterprise.exe --layout C:\vs2017offline --add Microsoft.VisualStudio.Workload.Azure --includeRecommended
```

- To download two workloads and one optional component for three languages, run:

```
vs_enterprise.exe --layout C:\vs2017offline --add Microsoft.VisualStudio.Workload.Azure --add Microsoft.VisualStudio.Workload.ManagedDesktop --add Component.GitHub.VisualStudio --includeRecommended --lang en-US de-DE ja-JP
```

- To download two workloads and all of their recommended components, run:

```
vs_enterprise.exe --layout C:\vs2017offline --add Microsoft.VisualStudio.Workload.Azure --add Microsoft.VisualStudio.Workload.ManagedDesktop --add Component.GitHub.VisualStudio --includeRecommended
```

- To download two workloads and all of their recommended and optional components, run:

```
vs_enterprise.exe --layout C:\vs2017offline --add Microsoft.VisualStudio.Workload.Azure --add Microsoft.VisualStudio.Workload.ManagedDesktop --add Component.GitHub.VisualStudio --includeOptional
```

New in 15.3

When you run a layout command, the options that you specify are saved (such as the workloads and languages).

Subsequent layout commands will include all of the previous options. Here is an example of a layout with one workload for English only:

```
vs_enterprise.exe --layout c:\VS2017Layout --add Microsoft.VisualStudio.Workload.ManagedDesktop --lang en-US
```

When you want to update that layout to a newer version, you don't have to specify any additional command-line parameters. The previous settings are saved and used by any subsequent layout commands in this layout folder. The following command will update the existing partial layout.

```
vs_enterprise.exe --layout c:\VS2017Layout
```

When you want to add an additional workload, here's an example of how to do so. In this case, we'll add the Azure workload and a localized language. Now, both Managed Desktop and Azure are included in this layout. The language resources for English and German are included for all these workloads. The layout is updated to the latest available version.

```
vs_enterprise.exe --layout c:\VS2017Layout --add Microsoft.VisualStudio.Workload.Azure --lang de-DE
```

If you want to update an existing layout to a full layout, use the `--all` option, as shown in the following example.

```
vs_enterprise.exe --layout c:\VS2017Layout --all
```

Deploying from a network installation

Administrators can deploy Visual Studio onto client workstations as part of an installation script. Or, users who have administrator rights can run setup directly from the share to install Visual Studio on their machine.

- Users can install by running:
`\server\products\VS2017\vs_enterprise.exe`
- Administrators can install in an unattended mode by running:
`\server\products\VS2017\vs_enterprise.exe --quiet --wait --norestart`

TIP

When executed as part of a batch file, the `--wait` option ensures that the `vs_enterprise.exe` process waits until the installation is complete before it returns an exit code. This is useful if an enterprise administrator wants to perform further actions on a completed installation (for example, to [apply a product key to a successful installation](#)) but must wait for the installation to finish to handle the return code from that installation. If you do not use `--wait`, the `vs_enterprise.exe` process exits before the installation is complete and returns an inaccurate exit code that doesn't represent the state of the install operation.

When you install from a layout, the content that is installed is acquired from the layout. However, if you select a component that is not in the layout, it will be acquired from the internet. If you want to prevent Visual Studio setup from downloading any content that is missing in your layout, use the `--noWeb` option. If `--noWeb` is used and the layout is missing any content that is selected to be installed, setup fails.

Error codes

If you used the `--wait` parameter, then depending on the result of the operation, the `%ERRORLEVEL%` environment variable is set to one of the following values:

| VALUE | RESULT |
|-------|----------------------------------|
| 0 | Operation completed successfully |

| VALUE | RESULT |
|-------|---|
| 3010 | Operation completed successfully, but install requires reboot before it can be used |
| Other | Failure condition occurred - check the logs for more information |

Updating a network install layout

As product updates become available, you might want to [update the network install layout](#) to incorporate updated packages.

How to create a layout for a previous Visual Studio 2017 release

NOTE

The Visual Studio 2017 bootstrappers that are available on [VisualStudio.com](#) download and install the latest Visual Studio 2017 release available whenever they are run. If you download a Visual Studio bootstrapper today and run it six months from now, it installs the Visual Studio 2017 release that is available at that later time. If you create a layout, installing Visual Studio from that layout installs the specific version of Visual Studio that exists in the layout. Even though a newer version might exist online, you get the version of Visual Studio that is in the layout.

If you need to create a layout for an older version of Visual Studio 2017, you can go to <https://my.visualstudio.com> to download "fixed" versions of the Visual Studio 2017 bootstrappers.

How to get support for your offline installer

If you experience a problem with your offline installation, we want to know about it. The best way to tell us is by using the [Report a Problem](#) tool. When you use this tool, you can send us the telemetry and logs we need to help us diagnose and fix the problem.

We have other support options available, too. For a list, see our [Talk to us](#) page.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio](#)
- [Visual Studio administrator guide](#)
- [Use command-line parameters to install Visual Studio](#)

- Visual Studio workload and component IDs

Update a network-based installation of Visual Studio

12/22/2017 • 6 min to read • [Edit Online](#)

It's possible to update a network installation layout of Visual Studio with the latest product updates so that it can be used both as an installation point for the latest update of Visual Studio and also to maintain installations that are already deployed to client workstations.

How to update a network layout

To refresh your network install share so that it includes the latest updates, run the --layout command to incrementally download updated packages.

If you selected a partial layout when you first created the network layout, those settings are saved. Any future layout commands use the previous options plus any new options that you specify. (This is new in 15.3.) If you are using a layout of an older version, you should use the same command-line parameters that you used when you first created the network install layout (in other words, the same workloads and languages) to update its content.

If you host a layout on a file share, you should update a private copy of the layout (for example, c:\vs2017offline) and then, after all of the updated content is downloaded, copy it to your file share (for example, \server\products\VS2017). If you don't do this, there is a greater chance that any users who run Setup while you are updating the layout might not be able to get all of the content from the layout because it is not yet completely updated.

Let's walk through how to create and then update a layout:

- First, here's an example of how to create a layout with one workload for English only:

```
vs_enterprise.exe --layout c:\VS2017Layout --add Microsoft.VisualStudio.Workload.ManagedDesktop --lang en-US
```

- Here's how to update that same layout to a newer version. You don't have to specify any additional command-line parameters. The previous settings were saved and will be used by any subsequent layout commands in this layout folder.

```
vs_enterprise.exe --layout c:\VS2017Layout
```

- Here's how to update your layout to a newer version in an unattended manner. The layout operation runs the setup process in a new console window. The window is left open so users can see the final result and a summary of any errors that might have occurred. If you are performing a layout operation in an unattended manner (for example, you have a script that is regularly run to update your layout to the latest version), then use the `--passive` parameter and the process will automatically close the window.

```
vs_enterprise.exe --layout c:\VS2017Layout --passive
```

- Here's how to add an additional workload and localized language. (This command adds the Azure workload.) Now both Managed Desktop and Azure are included in this layout. The language resources for English and German are also included for all these workloads. And, the layout is updated to the latest available version.

```
vs_enterprise.exe --layout c:\VS2017Layout --add Microsoft.VisualStudio.Workload.Azure --lang de-DE
```

- And finally, here's how to add an additional workload and localized language without updating the version. (This command adds the ASP.NET & Web workload.) Now the Managed Desktop, Azure, and ASP.NET & Web workloads are included in this layout. The language resources for English, German, and French are also included for all these workloads. However, the layout was not updated to the latest available version when this command was run. It remains at the existing version.

```
vs_enterprise.exe --layout c:\VS2017Layout --add Microsoft.VisualStudio.Workload.NetWeb --lang fr-FR --keepLayoutVersion
```

How to deploy an update to client machines

Depending on how your network environment is configured, an update can either be deployed by an enterprise administrator or initiated from a client machine.

- Users can update a Visual Studio instance that was installed from an offline installation folder:
 - Run the Visual Studio Installer.
 - Then, click **Update**.
- Administrators can update client deployments of Visual Studio without any user interaction with two separate commands:
 - First, update the Visual Studio installer:

```
vs_enterprise.exe --quiet --update
```
 - Then, update the Visual Studio application itself:

```
vs_enterprise.exe update --installPath "C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise" --quiet --wait --norestart
```

NOTE

Use the [vswhere.exe command](#) to identify the install path of an existing instance of Visual Studio on a client machine.

TIP

For details on how to control when update notifications are presented to users, see [Control updates to network-based Visual Studio deployments](#).

How to verify a layout

Use `--verify` to perform verification on the offline cache supplied. It checks if packages files are either missing or invalid. At the end of the verification, it prints the list of missing files and invalid files.

```
vs_enterprise.exe --layout <layoutDir> --verify
```

The vs_enterprise.exe can be invoked inside the layoutDir.

NOTE

Some important metadata files that are needed by the `--verify` option must be in the layout offline cache. If these metadata files are missing, "`--verify`" cannot run and Setup gives you an error. If you experience this error, re-create a new offline layout to a different folder (or to the same offline cache folder). To do so, run the same layout command that you used to create the initial offline layout. For example, `vs_enterprise.exe --layout <layoutDir>`.

Microsoft ships updates to Visual Studio periodically, so the new layout that you create might not be the same version as the initial layout.

How to fix a layout

Use `--fix` to perform the same verification as `--verify` and also try to fix the identified issues. The `--fix` process needs an Internet connection, so make sure your machine is connected to the Internet before you invoke `--fix`.

```
vs_enterprise.exe --layout <layoutDir> --fix
```

The `vs_enterprise.exe` can be invoked inside the `<layoutDir>`.

How to remove older versions from a layout

After you perform layout updates to an offline cache, the layout cache folder may have some obsolete packages that are no longer needed by the latest Visual Studio installation. You can use the `--clean` option to remove obsolete packages from an offline cache folder.

To do this, you'll need the file path(s) to catalog manifest(s) that contain those obsolete packages. You can find the catalog manifests in an "Archive" folder in the offline layout cache. They are saved there when you update a layout. In the "Archive" folder, there is one or more "GUID" named folders, each of which contains an obsolete catalog manifest. The number of "GUID" folders should be the same as the number of updates made to your offline cache.

A few files are saved inside each "GUID" folder. The two files of most interest are a "catalog.json" file and a "version.txt" file. The "catalog.json" file is the obsolete catalog manifest you'll need to pass to the `--clean` option. The other version.txt file contains the version of this obsolete catalog manifest. Based on the version number, you can decide whether you want to remove obsolete packages from this catalog manifest. You can do the same as you go through the other "GUID" folders. After you make the decision on the catalog(s) you want to clean, run the `--clean` command by supplying the files paths to these catalogs.

Here are a few examples of how to use the `--clean` option:

```
vs_enterprise.exe --layout <layoutDir> --clean <file-path-of-catalog1> <file-path-of-catalog2> ...
```

```
vs_enterprise.exe --layout <layoutDir> --clean <file-path-of-catalog1> --clean <file-path-of-catalog2> ...
```

You can also invoke `vs_enterprise.exe` inside the `<layoutDir>`. Here's an example:

```
c:\VS2017Layout\vs_enterprise.exe --layout c:\VS2017Layout --clean c:\VS2017Layout\Archive\1cd70189-fc55-4583-8ad8-a2711e928325\Catalog.json --clean c:\VS2017Layout\Archive\d420889f-6aad-4ba4-99e4-ed7833795a10\Catalog.json
```

When you execute this command, Setup analyzes your offline cache folder to find the list of files that it will

remove. You will then have a chance to review the files that are going to be deleted and confirm the deletions.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio](#)
- [Visual Studio administrator guide](#)
- [Use command-line parameters to install Visual Studio](#)
- [Tools for detecting and managing Visual Studio instances](#)
- [Control updates to network-based Visual Studio deployments](#)

Install and use Visual Studio and Azure Services behind a firewall or proxy server

2/15/2018 • 9 min to read • [Edit Online](#)

If you or your organization uses security measures such as a firewall or a proxy server, then there are domain URLs that you might want to "whitelist" and ports and protocols that you might want to open so that you have the best experience when you install and use Visual Studio and Azure Services.

- **Install Visual Studio:** These tables include the domain URLs to whitelist so that you have access to all the components and workloads that you want.
- **Use Visual Studio and Azure Services:** This table includes the domain URLs to whitelist and the ports and protocols to open so that you have access to all the features and services that you want.

Install Visual Studio

URLs to whitelist

Because the Visual Studio Installer downloads files from various domains and their download servers, here are the domain URLs that you might want to whitelist as trusted in the UI or in your deployment scripts.

Microsoft domains

| DOMAIN | PURPOSE |
|--|--|
| go.microsoft.com | Setup URL resolution |
| aka.ms | Setup URL resolution |
| download.visualstudio.microsoft.com | Setup packages download location |
| download.microsoft.com | Setup packages download location |
| download.visualstudio.com | Setup packages download location |
| dl.xamarin.com | Setup packages download location |
| visualstudiogallery.msdn.microsoft.com | Visual Studio Extensions download location |
| www.visualstudio.com | Documentation location |
| docs.microsoft.com | Documentation location |
| msdn.microsoft.com | Documentation location |
| www.microsoft.com | Documentation location |
| *.windows.net | Sign-in location |
| *.microsoftonline.com | Sign-in location |

| DOMAIN | PURPOSE |
|------------|------------------|
| *.live.com | Sign-in location |

Non-Microsoft domains

| DOMAIN | INSTALLS THESE WORKLOADS |
|-----------------------------------|--|
| archive.apache.org | Mobile development with JavaScript (Cordova) |
| cocos2d-x.org | Game development with C++ (Cocos) |
| download.epicgames.com | Game development with C++ (Unreal Engine) |
| download.oracle.com | Mobile development with JavaScript (Java SDK) Mobile Development with .NET (Java SDK) |
| download.unity3d.com | Game development with Unity (Unity) |
| netstorage.unity3d.com | Game development with Unity (Unity) |
| dl.google.com | Mobile development with JavaScript (Android SDK and NDK, Emulator) Mobile Development with .NET (Android SDK and NDK, Emulator) |
| www.incredibuild.com | Game development with C++ (Incredibuild) |
| incredibuildvs2017i.azureedge.net | Game development with C++ (Incredibuild) |
| www.python.org | Python development (Python) Data science and analytical applications (Python) |

Use Visual Studio and Azure Services

URLs to whitelist and ports and protocols to open

To make sure that you have access to everything you need when you use Visual Studio or Azure Services behind a firewall or proxy server, here are the URLs you should whitelist and the ports and protocols that you might want to open.

| SERVICE OR SCENARIO | DNS ENDPOINT | PROTOCOL | PORT | DESCRIPTION |
|---------------------|----------------------------|----------|------|---|
| URL resolution | go.microsoft.com aka.ms | | | Used to shorten URLs, which then resolve into longer URLs |

| Service or Scenario | DNS Endpoint | Protocol | Port | Description |
|--|---|----------|-----------|---|
| Start Page | vsstartpage.blob.core.windows.net | | 443 | Used to display Developer News shown on the start page in Visual Studio |
| Targeted Notification Service | targetednotifications.azurewebsites.net www.research.net | | 80 443 | Used to filter a global list of notifications to a list that is applicable only to specific types of machines/usage scenarios |
| Extension update check | visualstudiogallery.msdn.microsoft.com *.windows.net *.microsoftonline.com *.live.com | | 443 | Used to provide notifications when an installed extension has an update available Used as a sign-in location |
| AI Project Integration | az861674.vo.msecnd.net | | 443 | Used to configure new projects to send usage data to your registered Application Insights account |
| Code Lens | codelensprodscus1su0.app.codelens.visualstudio.com | | 443 | Used to provide information in the editor about when a file was last updated, the timeline of changes, the work items that changes are associated with, the authors, and more |
| Experimental feature enabling | visualstudio-devdiv-c2s.msedge.net | | 80 | Used to activate experimental new features or feature changes |
| Identity "badge" (user name and avatar) and Roaming settings | app.vssps.visualstudio.com app.vsspsext.visualstudio.com app.vssps.visualstudio.com ns-sb2-prod-ch1-002.cloudapp.net az700632.vo.msecnd.net | | 443 | Used to display the user's name and avatar in the IDE Used to make sure that setting changes roam from one machine to another |

| Service or Scenario | DNS Endpoint | Protocol | Port | Description |
|---|---|--------------------------|----------------------|---|
| Remote Settings | az700632.vo.msecnd.net | | 443 | Used to turn off extensions that are known to cause problems in Visual Studio |
| Windows Tools | developer.microsoft.com dev.windows.com appdev.microsoft.com | https | 443 | Used for Windows app store scenarios |
| JSON Schema Discovery | json.schemastore.org schemastoreorg.azurewebsites.net | http https | 80 443 | Used to discover and download JSON schemas that the user might use when editing JSON documents |
| JSON Schema Definition | json-schema.org | http | 443 | |
| JSON Schema Support for Azure Resources | schema.management.azure.com | https | 443 | Used to obtain the meta-validation schema for JSON |
| | | | | Used to obtain the current schema for Azure Resource Manager deployment templates |
| NPM package discovery | Skimdb.npmjs.com Registry.npmjs.org Api.npms.io | https http/s https | 443 80/443 443 | Required for searching for NPM packages, and used for client-side script package installation in web projects |
| Bower package icons | Bower.io | http | 80 | Provides the default bower package icon |
| Bower package search | bowercache.azurewebsites.net go.microsoft.com Registry.bower.io | https http https | 443 80 443 | Provides the ability to search for Bower packages |
| NuGet | Api.nuget.org www.nuget.org Nuget.org | https | 443 | Used to verify signed NuGet packages. |
| NuGet package discovery | crl3.digicert.com crl4.digicert.com ocsp.digicert.com cacerts.digicert.com | http/s | 80/443 | Required for searching for NuGet packages and versions |
| GitHub repository information | api.github.com | https | 443 | Required for getting additional information about bower packages |

| Service or Scenario | DNS Endpoint | Protocol | Port | Description |
|---|--|----------|------|--|
| Web Linters | Eslint.org www.Bing.com www.coffeelint.org | http | 80 | |
| Cookiecutter Explorer template discovery Cookiecutter Explorer project creation | api.github.com raw.githubusercontent.com go.microsoft.com pypi.org pypi.python.org | https | 443 | Used to discover online templates from our recommended feed and from github repositories Used to create a project from a cookiecutter template that requires a one-time on-demand installation of a cookiecutter Python package from the Python package index (PyPI) |
| Python package discovery Python package management Python New Project templates | pypi.org pypi.python.org bootstrap.pypa.io go.microsoft.com | https | 443 | Provides the ability to search for pip packages Used to install pip automatically if it is missing Used to create the Used to resolve the following Python project templates in the New Project dialog to cookiecutter template URLs: - Classifier Project - Clustering Project - Regression Project - PyGame using PyKinect - Pivot Project |
| Office web add-in Manifest Verification Service | verificationservice.osi.office.net | https | 443 | Used to validate manifests for Office web add-ins |
| SharePoint and Office Add-ins | sharepoint.com | https | 443 | Used to publish and test SharePoint and Office Add-ins to SharePoint Online |

| Service or Scenario | DNS Endpoint | Protocol | Port | Description |
|--|---|----------|-------|---|
| Workflow Manager Test Service Host | | http | 12292 | A firewall rule that is created automatically for testing SharePoint add-ins with workflows |
| Automatically collected reliability statistics and other Customer Experience Improvement Programs (CEIP) for Azure SDK and for SQL Tools | vortex.data.microsoft.com dc.services.visualstudio.com | https | 443 | Used to send reliability statistics (crash/hang data) from the user to Microsoft. Actual crash/hang dumps will still be uploaded if Windows Error Reporting is enabled; only statistical information will be suppressed; Used to reveal anonymous usage patterns for the Azure Tools SDK extension to Visual Studio, and for usage patterns for the SQL tooling to Visual Studio |
| Visual Studio Customer Experience Improvement Program (CEIP) PerfWatson.exe | vortex.data.microsoft.com dc.services.visualstudio.com visualstudio-devdiv-c2s.msedge.net az667904.vo.msecnd.net scus-breeziest-in.cloudapp.net | https | 443 | Used to collect anonymous usage patterns and error logs Used to track UI freeze issues |
| Creation and Management of Azure resources | management.azure.com management.core.windows.net | https | 443 | Used for creating Azure Websites or other resources to support the publishing of web applications, Azure Functions, or WebJobs |
| Updated web publish tooling checks and extension recommendations | marketplace.visualstudio.com visualstudiogallery.microsoft.com | https | 443 | Used for checking for the availability of updated publish tooling. If disabled, a potential recommended extension for web publishing may not be shown |

| Service or Scenario | DNS Endpoint | Protocol | Port | Description |
|---|---------------------------------------|----------|--------|--|
| Updated Azure Resource Creation Endpoint Information | *.blob.core.windows.net | https | 443 | Used to update the endpoints used for the creation of Azure Resources for certain Azure Services. If disabled, the last downloaded or built in endpoint locations are used instead |
| Remote debugging and Remote profiling of Azure Websites | *.cloudapp.net *.azurewebsites.net | | 4022 | Used for attaching the remote debugger to Azure Websites. If disabled, attaching the remote debugger to Azure Websites will not work |
| Active Directory Graph | graph.windows.net | https | 443 | Used to provision new Azure Active Directory applications. Also used by the Office 365 MSGraph-connected service provider |
| Azure Functions CLI Update Check | functionscdn.azureedge.net | https | 443 | Used for checking for updated versions of the Azure Functions CLI. If disabled, a cached copy (or the copy carried by the Azure Functions component) of the CLI will be used instead |
| Cordova | npmjs.org gradle.org | http/s | 80/443 | HTTP is used for Gradle downloads during build; HTTPS is used to include Cordova plug-ins in projects |

| Service or Scenario | DNS Endpoint | Protocol | Port | Description |
|---------------------|---|--|--|--|
| Cloud explorer | 1. <clusterendpoint> Service Fabric 2. <management endpoint> General Cloud Exp 3. <graph endpoint> General Cloud Exp 4. <storage account endpoint> Storage Nodes 5. <Azure portal URLs> General Cloud Exp 6. <key vault endpoints> Azure Resource Manager VM Nodes 7. <PublicIPAddressOfCluster> Service Fabric Remote debugging and ETW Traces | 1. https 2. https 3. https 4. https 5. https 6. https 7: tcp | 1. 19080 2. 443 3. 443 4. 443 5. 443 6. 443 7. dynamic | <p>1. Example: test12.eastus.cloudapp.com</p> <p>2. Retrieves subscriptions and retrieves/manages Azure resources</p> <p>3. Retrieves Azure Stack subscriptions</p> <p>4. Manages Storage resources (example: mystorageaccount.blob.core.windows.net)</p> <p>5. "Open in Portal" context menu option (opens a resource in the Azure portal)</p> <p>6. Creates and uses key vaults for VM debugging (Example: myvault.vault.azure.net)</p> <p>7. Dynamically allocates block of ports based on number of nodes in the cluster and the available ports.</p> <p>A port block will try to get three times the number of nodes with minimum of 10 ports.</p> <p>For Streaming traces, an attempt is made to get the port block from 810. If any of that port block is already used, then an attempt is made to get the next block, and so on. (If the load balancer is empty, then ports from 810 are most likely used)</p> <p>Similarly for debugging, four sets of the ports blocks are reserved:</p> <ul style="list-style-type: none"> - connectorPort: 30398, - forwarderPort: 31398, - forwarderPortx86: 31399, - fileUploadPort: 32398 |

| Service or Scenario | DNS Endpoint | Protocol | Port | Description |
|---------------------|--|--|---|--|
| Cloud Services | 1. RDP 2. core.windows.net 3. management.azure.com management.core.windows.net 4. *.blob.core.windows.net *.queue.core.windows.net *.table.core.windows.net 5. portal.azure.com 6. <user's cloud service>.cloudapp.net <user's VM>. <region>.azure.com | 1. rdp 2. https 3. https 4. https 5. https 6. tcp | 1. 3389 2. 443 3. 443 4. 443 5. 443 6. a) 30398 6. b) 30400 6. c) 31398 6. d) 31400 6. e) 32398 6. f) 32400 | 1. Remote Desktop to Cloud Services VM 2. Storage account component of the private diagnostics configuration 3. Azure portal 4. Server Explorer - Azure Storage * is customer named storage account 5. Links to open the portal / Download the subscription certificate / Publish settings file 6. a) Connector local port for remote debug for cloud service and VM 6. b) Connector public port for remote debug for cloud service and VM 6. c) Forwarder local port for remote debug for cloud service and VM 6. d) Forwarder public port for remote debug for cloud service and VM 6. e) File uploader local port for remote debug for cloud service and VM 6. f) File uploader public port for remote debug for cloud service and VM |

| Service or Scenario | DNS Endpoint | Protocol | Port | Description |
|---|--|---|--|---|
| Service Fabric | 1. ocs.Microsoft.com aka.ms go.microsoft.com 2. vssftools.blob.core.windows.net Vault.azure.com Portal.azure.com 3. * vault.azure.net 4. app.vsaex.visualstudio.com * .vsspsext.visualstudio.com clouds.vsrn.visualstudio.io.com clouds.visualstudio.com app.vssps.visualstudio.com *.visualstudio.com | https | 443 | 1. Documentation 2. Create Cluster feature 3. The * is the Azure key vault name (Example:- test11220180112110 108.vault.azure.net) 4. The * is dynamic (Example: vsspsextprodch1su1.vsspsext.visualstudio.com) |
| Snapshot Debugger | 1. go.microsoft.com 2. management.azure.com 3. *azurewebsites.net 4. *scm.azurewebsites.net 5. api.nuget.org/v3/index.json 6. msvsmon | 1. https 2. https 3. http 4. https 5. https 6. Concord | 1. 443 2. 443 3. 80 4. 443 5. 443 6. 4022 (Visual Studio version dependent) | 1. Query .json file for app service SKU size 2. Various Azure RM calls 3. Site warmup call via 4. Customer's targeted App Service Kudu endpoint 5. Query Site Extension version published in nuget.org 6. Remote debugging channel |
| Azure Stream Analytics HDInsight | Management.azure.com | https | 443 | Used to view, submit, run, and manage ASA jobs Used to browse HDI clusters, and to submit, diagnose, and debug HDI jobs |
| Azure Data Lake | *.azuredatalakestore.net *.azuredatalakeanalyticstcs.net | https | 443 | Used to compile, submit, view, diagnose, and debug jobs; used to browse ADLS files; used to upload and download files |
| | | | | |

Troubleshoot network-related errors

Sometimes, you might run in to network- or proxy-related errors when you install or use Visual Studio behind a firewall or a proxy server. For more information about solutions for such error messages, see the [Troubleshooting network-related errors when you install or use Visual Studio](#) page.

Get support

Here are a few more support options for you:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Troubleshooting network-related errors in Visual Studio](#)
- [Visual Studio Administrator Guide](#)
- [Install Visual Studio 2017](#)

Troubleshooting network-related errors when you install or use Visual Studio

2/15/2018 • 3 min to read • [Edit Online](#)

We've got solutions for the most typical network- or proxy-related errors that you might encounter when you install or use Visual Studio behind a firewall or a proxy server.

Error: "Proxy authorization required"

This error generally occurs when users are connected to the internet through a proxy server, and the proxy server blocks the calls that Visual Studio makes to some network resources.

To fix this error:

- Restart Visual Studio. A proxy authentication dialog box should appear. Enter your credentials when prompted in the dialog.
- If restarting Visual Studio does not solve the problem, it might be that your proxy server does not prompt for credentials for `http://go.microsoft.com` addresses but does so for `*.visualStudio.com` addresses. For these servers, consider whitelisting the following URLs to unblock all sign-in scenarios in Visual Studio:
 - `*.windows.net`
 - `*.microsoftonline.com`
 - `*.visualstudio.com`
 - `*.microsoft.com`
 - `*.live.com`

- You can otherwise remove the `http://go.microsoft.com` address from the whitelist so that the proxy authentication dialog shows up for both the `http://go.microsoft.com` address and the server endpoints when Visual Studio is restarted.

OR

- If you want to use your default credentials with your proxy, you can perform the following actions:
 1. Find **devenv.exe.config** (the devenv.exe configuration file) in: **%ProgramFiles%\Microsoft Visual Studio\2017\Enterprise\Common7\IDE** or **%ProgramFiles(x86)%\Microsoft Visual Studio\2017\Enterprise\Common7\IDE**.
 2. In the configuration file, find the `<system.net>` block, and then add this code:

```
<defaultProxy enabled="true" useDefaultCredentials="true">
    <proxy bypassOnLocal="True" proxyAddress=" HYPERLINK "http://<yourproxy:port#>
    http://<yourproxy:port#>" />
</defaultProxy>
```

You must insert the correct proxy address for your network in
`proxyAddress="<http://<yourproxy:port#> .`

OR

- You can also follow the instructions in the [How to connect through an authenticated Web Proxy](#) blog post, which shows you how to add code that will allow you to use the proxy.

Error: “The underlying connection was closed”

If you are using Visual Studio in a private network that has a firewall, Visual Studio might not be able to connect to some network resources. These resources can include Visual Studio Team Services (VSTS) for sign-in and licensing, NuGet, and Azure services. If Visual Studio fails to connect to one of these resources, you might see the following error message:

The underlying connection was closed: An unexpected error occurred on send

Visual Studio uses Transport Layer Security (TLS) 1.2 protocol to connect to network resources. Security appliances on some private networks block certain server connections when Visual Studio uses TLS 1.2.

To fix this error:

Enable connections for the following URLs:

- <https://management.core.windows.net>
- <https://app.vssps.visualstudio.com>
- <https://login.microsoftonline.com>
- <https://login.live.com>
- <https://go.microsoft.com>
- <https://graph.windows.net>
- <https://app.vsspsext.visualstudio.com>
- *.azurewebsites.net (for Azure connections)
- *.visualstudio.com
- cdn.vsassets.io (hosts content delivery network, or CDN, content)
- *.gallerycdn.vsassets.io (hosts VSTS extensions)
- static2.sharepointonline.com (hosts resources that Visual Studio uses in the Office UI Fabric kit, such as fonts)
- *.nuget.org (for NuGet connections)

NOTE

Privately owned NuGet server URLs may not be included in this list. You can check for the NuGet servers that you are using in %APPData%\Nuget\NuGet.Config.

Get support

If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the installation troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.

- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install and use Visual Studio behind a firewall or proxy server](#)
- [Visual Studio Administrator Guide](#)
- [Install Visual Studio 2017](#)

How to define settings in a response file

12/22/2017 • 3 min to read • [Edit Online](#)

Administrators who deploy Visual Studio can specify a response file by using the `--in` parameter, as in the following example:

```
vs_enterprise.exe --in customInstall.json
```

Response files are [JSON](#) files whose contents mirror the command-line arguments. In general, if a command-line parameter takes no arguments (for example, `--quiet`, `--passive`, etc.), the value in the response file should be true/false. If it takes an argument (for example, `--installPath <dir>`), the value in the response file should be a string. If it takes an argument and can appear on the command line more than once (for example, `--add <id>`), it should be an array of strings.

Parameters that are specified on the command-line override settings from the response file, except when parameters take multiple inputs (for example, `--add`). When you have multiple inputs, the inputs supplied on the command line are merged with settings from the response file.

Setting a default configuration for Visual Studio

If you created a network layout cache with the `--layout`, an initial `response.json` file is created in the layout. If you create a partial layout, this response file includes the workloads and languages that were included in the layout. Running setup from this layout automatically uses this `response.json` file, which selects the workloads and components included in the layout. Users can still select or unselect any workloads in the setup UI before installing Visual Studio.

Administrators who create a layout can modify the `response.json` file in the layout to control the default settings that their users see when they install Visual Studio from the layout. For example, if an administrator wants specific workloads and components installed by default, they can configure the `response.json` file to add them.

When Visual Studio setup is run from a layout folder, it *automatically* uses the response file in the layout folder. You don't have to use the `--in` option.

You can update the `response.json` file that is created in an offline layout folder to define the default setting for users who install from this layout.

WARNING

It's critical that you leave the existing properties that were defined when the layout was created.

The base `response.json` file in a layout should look similar to the following example, except that it would include the value for the product and channel that you want to install:

```
{
  "installChannelUri": ".\\ChannelManifest.json",
  "channelUri": "https://aka.ms/vs/15/release/channel",
  "installCatalogUri": ".\\Catalog.json",
  "channelId": "VisualStudio.15.Release",
  "productId": "Microsoft.VisualStudio.Product.Enterprise"
}
```

When you create or update a layout, a response.template.json file is also created. This file contains all of the workload, component, and language IDs that can be used. This file is provided as a template for what all could be included in a custom install. Administrators can use this file as a starting point for a custom response file. Just remove the IDs for the things you do not want to install and save it in your own response file. Do not customize the response.template.json file or your changes will be lost whenever the layout is updated.

Example layout response file content

The following example installs Visual Studio Enterprise with six common workloads and components, and with both English and French UI languages. You can use this example as a template; just change the workloads and components to those that you want to install:

```
{  
  "installChannelUri": ".\\ChannelManifest.json",  
  "channelUri": "https://aka.ms/vs/15/release/channel",  
  "installCatalogUri": ".\\Catalog.json",  
  "channelId": "VisualStudio.15.Release",  
  "productId": "Microsoft.VisualStudio.Product.Enterprise",  
  
  "installPath": "C:\\VS2017",  
  "quiet": false,  
  "passive": false,  
  "includeRecommended": true,  
  "norestart": false,  
  
  "addProductLang": [  
    "en-US",  
    "fr-FR"  
  ],  
  
  "add": [  
    "Microsoft.VisualStudio.Workload.ManagedDesktop",  
    "Microsoft.VisualStudio.Workload.Data",  
    "Microsoft.VisualStudio.Workload.NativeDesktop",  
    "Microsoft.VisualStudio.Workload.NetWeb",  
    "Microsoft.VisualStudio.Workload.Office",  
    "Microsoft.VisualStudio.Workload.Universal",  
    "Component.GitHub.VisualStudio"  
  ]  
}
```

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Visual Studio 2017 workload and component IDs](#)

Automatically apply product keys when deploying Visual Studio

12/22/2017 • 2 min to read • [Edit Online](#)

You can apply your product key programmatically as part of a script that is used to automate the deployment of Visual Studio. You can set a product key on a device programmatically either during an installation of Visual Studio or after an installation completes.

Apply the license after installation

You can activate an installed version of Visual Studio with a product key by using the `StorePID.exe` utility on the target machines, in silent mode. `StorePID.exe` is a utility program that installs with Visual Studio 2017 at the following default location:

```
C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\Common7\IDE
```

Run `StorePID.exe` with elevated privileges, either by using a System Center agent or an elevated command prompt. Follow it with the product key and the Microsoft Product Code (MPC).

IMPORTANT

Make sure to include the dashes in the product key.

```
StorePID.exe [product key including the dashes] [MPC]
```

The following example shows a command line for applying the license for Visual Studio 2017 Enterprise, which has an MPC of 08860, a product key of `AAAAAA-BBBBBB-CCCCCC-DDDDDD-EEEEEEE`, and assumes a default installation location:

```
"C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\Common7\IDE\StorePID.exe" AAAAAA-BBBBBB-CCCCCC-DDDDDD-EEEEEEE 08860
```

The following table lists the MPC codes for each edition of Visual Studio:

| VISUAL STUDIO EDITION | MPC |
|--------------------------------------|-------|
| Visual Studio Enterprise 2017 | 08860 |
| Visual Studio Professional 2017 | 08862 |
| Visual Studio Test Professional 2017 | 08866 |

If `StorePID.exe` successfully applies the product key, it returns an `%ERRORLEVEL%` of 0. If it encounters errors, it returns one of the following codes, depending on the error condition:

| ERROR | CODE |
|---------------------------------|------|
| <code>PID_ACTION_SUCCESS</code> | 0 |

| ERROR | CODE |
|-------------------------|------|
| PID_ACTION_NOTINSTALLED | 1 |
| PID_ACTION_INVALID | 2 |
| PID_ACTION_EXPIRED | 3 |
| PID_ACTION_INUSE | 4 |
| PID_ACTION_FAILURE | 5 |
| PID_ACTION_NOUPGRADE | 6 |

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See Also

- [Install Visual Studio](#)
- [Create an offline installation of Visual Studio](#)

Set defaults for enterprise deployments of Visual Studio

3/21/2018 • 2 min to read • [Edit Online](#)

You can set registry policies that affect the deployment of Visual Studio. These policies are global for the new installer and affect:

- Where some packages shared with other versions or instances are installed
- Where packages are cached
- Whether all packages are cached

You can set some of these policies using [command-line options](#), set registry values on your machine, or even distribute them using Group Policy across an organization.

Registry keys

There are several locations where you can set enterprise defaults, to enable their control either through Group Policy or directly in the registry. Visual Studio looks sequentially to see if any enterprise policies have been set; as soon as a policy value is discovered in the order below, the remaining keys are ignored.

1. `HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\VisualStudio\Setup`
2. `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\Setup`
3. `HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\VisualStudio\Setup` (on 64-bit operating systems)

IMPORTANT

If you do not set the `HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\VisualStudio\Setup` key and instead set one of the other keys, you should set both other keys on 64-bit operating systems. This issue is addressed in a future product update.

Some registry values are set automatically the first time they are used if not set already. This practice ensures that subsequent installs use the same values. These values are stored in the second registry key,

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\Setup`.

You can set the following registry values:

| NAME | TYPE | DEFAULT | DESCRIPTION |
|------------------------|--|--|---|
| <code>CachePath</code> | <code>REG_SZ</code> or <code>REG_EXPAND_SZ</code> | <code>%ProgramData%\Microsoft\VisualStudio\Packages</code> | The directory where package manifests and, optionally, payloads are stored. Read how to disable or move the package cache for more information. |

| NAME | TYPE | DEFAULT | DESCRIPTION |
|------------------------|-------------------------|--|---|
| KeepDownloadedPayloads | REG_DWORD | 1 | Keep package payloads even after they are installed. You can change the value anytime. Disabling the policy removes any cached package payloads for the instance you repair or modify. Read how to disable or move the package cache for more information. |
| SharedInstallationPath | REG_SZ or REG_EXPAND_SZ | %ProgramFiles(x86)%\Microsoft Visual Studio\Shared | The directory where some packages shared across versions of instances of Visual Studio are installed. You can change the value anytime, but that will only affect future installs. Any products already installed to the old location must not be moved or they may not function correctly. |

IMPORTANT

If you change the `CachePath` registry policy after any installs you must move the existing package cache to the new location and make sure it's secured so that `SYSTEM` and `Administrators` have Full Control and `Everyone` has Read access. Failure to move the existing cache or securing it may cause problems with future installs.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio](#)
- [Disable or move the package cache](#)
- [Use command-line parameters to install Visual Studio](#)

Disable or move the package cache

3/21/2018 • 2 min to read • [Edit Online](#)

The package cache provides a source of installed packages in case you need to repair Visual Studio or other related products in cases where you have no Internet connection. With some drives or system set ups, however, you may not want to keep all those packages around. The installer will download them when needed, so if you want to save or recover disk space you can disable or move the package cache.

Disable the package cache

Before you install, modify, or repair Visual Studio or other products with the new installer, you can start the installer with the `--nocache` switch to the installer.

```
"%ProgramFiles(x86)%\Microsoft Visual Studio\Installer\vs_installer.exe" --nocache
```

Any operation you do on any product will remove any existing packages for that product and will avoid saving any packages after they are installed. If you modify or repair Visual Studio and packages are required, they will be downloaded automatically and removed after they are installed.

If you want to re-enable the cache, pass `--cache` instead. Only packages that are required will be cached, so if you need to restore all packages you should repair Visual Studio before you disconnect from your network.

```
"%ProgramFiles(x86)%\Microsoft Visual Studio\Installer\vs_installer.exe" repair --passive --norestart --cache
```

You can also set the `KeepDownloadedPayloads` [registry policy](#) to disable the cache before you install, modify, or repair Visual Studio.

Move the package cache

A common system configuration is to have Windows installed on an SSD with a larger hard disk (or more) for development needs, such as source code, program binaries, and more. If you want to work offline you can instead move the package cache.

Currently, you can only do this if you set the `CachePath` [registry policy](#) before you install, modify, or repair Visual Studio.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Microsoft Q&A forum](#).

[Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio](#)
- [Set defaults for enterprise deployments](#)
- [Use command-line parameters to install Visual Studio](#)

Control updates to network-based Visual Studio deployments

12/22/2017 • 3 min to read • [Edit Online](#)

Enterprise administrators often create a layout and host it on a network file share to deploy to their end-users.

Controlling where Visual Studio looks for updates

By default, Visual Studio continues to look online for updates even if the installation was deployed from a network share. If an update is available, the user can install it. Any updated content that is not found in the offline layout is downloaded from the web.

If you want direct control over where Visual Studio looks for updates, you can modify the location where it looks. You can also control the version your users are updated to. To do so, follow these steps:

1. Create an offline layout: `vs_enterprise.exe --layout C:\vs2017offline --lang en-US`
2. Copy it to the file share where you want to host it: `xcopy /e C:\vs2017offline \\server\share\VS2017`
3. Modify the response.json file in the layout and change the `channelUri` value to point to a copy of the channelManifest.json that the admin controls.

Be sure to escape backslashes in the value, as in the following example:

```
"channelUri": "\\\server\\share\\VS2017\\ChannelManifest.json"
```

Now end-users can run setup from this share to install Visual Studio.

```
\\server\share\VS2017\vs_enterprise.exe
```

When an enterprise administrator determines it is time for their users to update to a newer version of Visual Studio, they can [update the layout location](#) to incorporate the updated files, as follows.

1. Use a command that is similar to the following command:
`vs_enterprise.exe --layout \\server\share\VS2017 --lang en-US`
2. Ensure that the response.json file in the updated layout still contains your customizations, specifically the `channelUri` modification, as follows:

```
"channelUri": "\\\server\\share\\VS2017\\ChannelManifest.json"
```

Existing Visual Studio installs from this layout look for updates at `\\server\share\VS2017\ChannelManifest.json`. If the channelManifest.json is newer than what the user has installed, Visual Studio notifies the user that an update is available.

New installs automatically install the updated version of Visual Studio directly from the layout.

Controlling notifications in the Visual Studio IDE

As described earlier, Visual Studio checks the location from which it has been installed, such as a network share or the internet, to see whether any updates are available. When an update is available, Visual Studio notifies the user

with a notification flag in the top right-hand corner of the window.



You can disable the notifications if you don't want end-users to be notified of updates. (For example, you might want to disable notifications if you deliver updates through a central software distribution mechanism.)

Because Visual Studio 2017 [stores registry entries in a private registry](#), you can't directly edit the registry in the typical way. However, Visual Studio includes a `vsregedit.exe` utility that you can use to change Visual Studio settings. You can turn off notifications with the following command:

```
vsregedit.exe set "C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise" HKCU ExtensionManager  
AutomaticallyCheckForUpdates2Override dword 0
```

(Make sure to replace the directory to match the installed instance that you want to edit.)

TIP

Use `vswhere.exe` to find a specific instance of Visual Studio on a client workstation.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Visual Studio](#)
- [Visual Studio administrator guide](#)
- [Use command-line parameters to install Visual Studio](#)
- [Tools for managing Visual Studio instances](#)

Tools for detecting and managing Visual Studio instances

12/22/2017 • 2 min to read • [Edit Online](#)

Detecting existing Visual Studio instances

We have made several tools available that will help you detect and manage installed Visual Studio instances on client machines:

- [VSWhere](#): an executable built into Visual Studio or available for separate distribution that helps you find the location of all Visual Studio instances on a particular machine.
- [VSSetup.PowerShell](#): PowerShell scripts that use the Setup Configuration API to identify installed instances of Visual Studio.
- [VS-Setup-Samples](#): C# and C++ samples that demonstrate how to use the Setup Configuration API to query an existing installation.

In addition, the [Setup Configuration API](#) provides interfaces for developers who want to build their own utilities for interrogating Visual Studio instances.

Using vswhere.exe

`vswhere.exe` is automatically included in Visual Studio 2017 version 15.2 or above, or you may download it from [the releases page](#). Use `vswhere -?` to get help information about the tool. As an example, this command shows all releases of Visual Studio, including old versions of the product and prereleases, and outputs the results in JSON format:

```
C:\Program Files (x86)\Microsoft Visual Studio\Installer> vswhere.exe -legacy -prerelease -format json
```

TIP

For more information about Visual Studio 2017 installation, see [Heath Stewart's blog articles](#).

Editing the registry for a Visual Studio instance

In Visual Studio 2017, registry settings are stored in a private location, which enables multiple side-by-side instances of the same version of Visual Studio on the same machine.

As these entries are not stored in the global registry, there are special instructions for using the Registry Editor to make changes to registry settings:

1. If you have an open instance of Visual Studio 2017, close it.
2. Start `regedit.exe`.
3. Select the `HKEY_LOCAL_MACHINE` node.
4. From the Regedit main menu, select **File -> Load Hive...** and then select the private registry file, which is stored in the **AppData\Local** folder. For example:

```
%localappdata%\Microsoft\VisualStudio\<config>\privateregistry.bin
```

NOTE

`<config>` corresponds to the instance of Visual Studio that you would like to browse.

You will be prompted to provide a hive name, which becomes the name of your isolated hive. After you do so, you should be able to browse the registry under the isolated hive that you created.

IMPORTANT

Before you start Visual Studio again, you must unload the isolated hive that you created. To do this, select File -> Unload Hive from the Regedit main menu. (If you do not do this, then the file remains locked and Visual Studio will not be able to start.)

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Visual Studio Administrators Guide](#)

Help Viewer administrator guide

12/22/2017 • 3 min to read • [Edit Online](#)

The Help Viewer allows you to manage local Help installations for network environments with or without internet access. Local help content is configured on a per machine basis. By default, users must have administrator rights to update their local Help installation.

If your network environment allows clients to access the internet, you can use the Help Content Manager executable to deploy local Help content from the internet. For more information about HlpCtnMgr.exe command line syntax, see [Command-Line Arguments for the Help Content Manager](#).

For information about creating content, creating an intranet service endpoint, and similar types of activities, see the [Help Viewer SDK](#).

If do not have internet access in your network environment, Help Viewer can deploy local Help content from the intranet or a network share. You can also disable Visual Studio IDE Help options by using [registry key overrides](#) for functionality such as:

- online versus offline help
- content installation at first launch of the IDE
- specifying an intranet content service
- managing content

Deploying local Help content from the internet

You can use Help Content Manager (HlpCtnMgr.exe) to deploy local Help content from the internet to client computers. Use the following syntax:

```
\%ProgramFiles(x86)%\Microsoft Help Viewer\v2.3\HlpCtnmgr.exe /operation \<*name*> /catalogname \<*catalog name*> /locale \<*locale*>
```

For more information about HlpCtnMgr.exe command line syntax, see [Command-Line Arguments for the Help Content Manager](#).

Requirements:

- Client computers must have access to the internet.
- Users must have administrator rights to update, add, or remove the local Help content after it has been installed.

Caveats:

- The default source for Help will still be online.

Example

The following example installs English content for Visual Studio to a client computer.

To install English content from the internet

1. Choose **Start** and then choose **Run**.
2. Type the following:

```
C:\Program Files (x86)\Microsoft Help Viewer\v2.3\hlpcntmgr.exe /operation install /catalogname  
VisualStudio15 /locale en-us
```

3. Press **Enter**.

Deploying pre-installed local Help content on client computers

You can install a set of content from online to one computer, and then copy that installed set of content to other computers.

Requirements:

- The computer you install the set of content to must have access to the internet.
- Users must have administrator rights to update, add, or remove the local Help content after it has been installed.

TIP

If users do not have administrator rights, it is recommended that you disable the Manage Content tab in the Help Viewer. For more information, see [Help Content Manager Overrides](#).

Caveats:

- The default source for Help will still be online.

Create the content set

Before you can create the base content set, you must first uninstall all local Visual Studio content on the target computer.

To uninstall local help

1. In the Help Viewer, choose the **Manage Content** tab.
2. Navigate to the Visual Studio document set.
3. Choose **Remove** next to each sub-item.
4. Choose **Update** to uninstall.
5. Browse to %ProgramData%\Microsoft\HelpLibrary2\Catalogs\VisualStudio15 and verify that the folder only contains the file catalogType.xml.

Once you have removed all previously installed local Visual Studio Help content, you are ready to download the base content set.

To download the content

1. In the Help Viewer, choose the **Manage Content** tab.
2. Under **Recommended Documentation** or **Available Documentation**, navigate to the documentation sets you want to download and then choose **Add**.
3. Choose **Update**.

Next, you need to package the content so it can be deployed to client computers.

To package the content

1. Create a folder to copy the content to for later deployment. For example: C:\VSHelp.
2. Open cmd.exe with Administrator permissions.

3. Navigate to the folder you created in step 1.

4. Type the following:

```
Xcopy %ProgramData%\Microsoft\HelpLibrary2 <foldername>\ /y /e /k /o
```

For example: `Xcopy %ProgramData%\Microsoft\HelpLibrary2 c:\VSHelp\ /y /e /k /o`

Deploying the content

To deploy the content

1. Create a network share and copy the help content to that location.

For example, copy the content in C:\VSHelp to \\myserver\VSHelp.

2. Create a .bat file to contain the deployment script for the help content. Since the client could possibly have a read lock on any of the files being deleted as part of the push, you should have the client shut down prior to pushing updates. For example:

```
REM - copy pre-ripped content to ProgramData
Xcopy %~dp0HelpLibrary2 %SYSTEMDRIVE%\ProgramData\Microsoft\HelpLibrary2\ /y /e /k /o
if ERRORLEVEL 1 ECHO *** ERROR COPYING Help Library files to ProgramData (%ERRORLEVEL%)
```

3. Run the .bat file on the local machines that you want to install the Help content on.

See also

[Command-Line Arguments for the Help Content Manager](#)

[Help Content Manager Overrides](#)

[Microsoft Help Viewer](#)

[Help Viewer SDK](#)

Command-Line Arguments for the Help Content Manager

12/22/2017 • 5 min to read • [Edit Online](#)

You can specify how to deploy and manage local Help content by using command-line arguments for Help Content Manager (HlpCtnMgr.exe). You must run scripts for this command-line tool with administrator permissions, and you can't run these scripts as a service. You can perform the following tasks by using this tool:

- Add or update local Help content from a disk or the cloud.
- Remove local Help content.
- Move the local Help content store.
- Add, update, remove, or move local Help content silently.

Syntax:

```
HlpCtnMgr.exe /operation Value /catalogname CatalogName /locale Locale /sourceuri InstallationPoint
```

For example:

```
hlpctntmgr.exe /operation install /catalogname VisualStudio15 /locale en-us /sourceuri  
d:\productDocumentation\HelpContentSetup.msha
```

Switches and Arguments

The following table defines the switches and arguments that you can use for the command-line tool for Help Content Manager:

| SWITCH | REQUIRED? | ARGUMENTS |
|--------|-----------|-----------|
|--------|-----------|-----------|

| SWITCH | REQUIRED? | ARGUMENTS |
|--------------|-----------|--|
| /operation | Yes | <ul style="list-style-type: none"> - Install--Adds books from the specified installation source to the local content store. <p>This switch requires the /booklist argument, the /sourceURI argument, or both. If you don't specify the /sourceURI argument, the default Visual Studio URI is used as the installation source. If you don't specify the /booklist argument, all books on the /sourceUri are installed.</p> <ul style="list-style-type: none"> - Uninstall--Removes the books that you specify from the local content store. <p>This switch requires the /booklist argument or the /sourceURI argument. If you specify the /sourceURI argument, all books are removed, and the /booklist argument is ignored.</p> <ul style="list-style-type: none"> - Move--Moves the local store to the path that you specify. The default local store path is set as a directory under %ProgramData% <p>This switch requires the /locationPath and /catalogName arguments. Error messages will be logged in the event log if you specify a path that isn't valid or if the drive doesn't contain enough free space to hold the content.</p> <ul style="list-style-type: none"> - Refresh--Updates topics that have changed since they were installed or most recently updated. <p>This switch requires the /sourceURI argument.</p> |
| /catalogName | Yes | Specifies the name of the content catalog. |
| /locale | No | <p>Specifies the product locale that's used to view and manage content for the current instance of the Help viewer. For example, you specify <code>EN-US</code> for English-United States.</p> <p>If you don't specify a locale, the locale of the operating system is used. If that locale can't be determined, <code>EN-US</code> is used.</p> <p>If you specify a locale that isn't valid, an error message is logged in the event log.</p> |
| /e | No | Elevates the Help Content Manager to Administrative privileges if the current user has administrative credentials. |

| SWITCH | REQUIRED? | ARGUMENTS |
|--------------|-----------|--|
| /sourceURI | No | <p>Specifies the URL from which content is installed (Service API) or the path to the content installation file (.msha). The URL can point to the Product Group (top-level node) or to the Product Books (leaf-level node) in a Visual Studio 2010 style endpoint. You don't need to include a slash (/) at the end of the URL. If you do include a trailing slash, it will be handled appropriately.</p> <p>An error message is logged in the event log if you specify a file that isn't found, isn't valid, or isn't accessible or if a connection to the Internet isn't available or is interrupted while content is being managed.</p> |
| /vendor | No | Specifies the vendor for the product content that will be removed (for example, Microsoft). The default argument for this switch is Microsoft. |
| /productName | No | Specifies the product name for the books that will be removed. The product name is identified in the helpcontentsetup.msha or books.html files that shipped with the content. You can remove books from only one product at a time. To remove books from multiple products, you must perform multiple installations. |
| /booklist | No | <p>Specifies the names of the books to be managed, separated by spaces. Values must match the book names as listed on the installation media.</p> <p>If you don't specify this argument, all recommended books for the specified product in the /sourceURI are installed.</p> <p>If the name of a book contains one or more spaces, surround it with double quotes ("") so that the list is delimited appropriately.</p> <p>Error messages will be logged if you specify a /sourceURI that isn't valid or isn't reachable.</p> |
| /skuid | No | Specifies the stock keeping unit (SKU) of the product from the installation source, and filters books that the /SourceURI switch identifies. |

| SWITCH | REQUIRED? | ARGUMENTS |
|---------------|-----------|---|
| /membership | No | <ul style="list-style-type: none"> - Minimum-- Installs a minimum set of Help content based on the SKU that you specify by using the /skuld switch. The mapping between the SKU and the content set is exposed in the Service API. - Recommended--Installs a set of recommended books for the SKU that you specify by using the /skuld argument. The Installation source is the service API or .MSHA. - Full-- Installs the entire set of books for the SKU that you specify by using the /skuld argument. The Installation source is the service API or .MSHA. |
| /locationpath | No | Specifies the default folder for local Help content. You must use this switch only to install or move content. If you specify this switch, you must also specify the /silent switch. |
| /silent | No | Installs or removes Help content without prompting the user or displaying any UI, including the icon in the status notification area. Output is logged to a file in the %Temp% directory. Important: To install content silently, you must use digitally signed .cab files, not .mshc files. |
| /launchingApp | No | <p>Defines the application and catalog context when the Help viewer is launched without the parent application. The arguments for this switch are <i>CompanyName</i>, <i>ProductName</i>, and <i>VersionNumber</i> (for example,</p> <div style="border: 1px solid black; padding: 5px; display: inline-block;"> <pre>/launchingApp Microsoft,VisualStudio,15.0</pre> </div> <p>).</p> <p>This is required for installing content with the /silent parameter."</p> |
| /wait Seconds | No | Pauses install, uninstall, and refresh operations. If an operation is already in progress for the catalog, the process will wait up to the given number of seconds to continue. Use 0 to wait indefinitely. |
| /? | No | Lists the switches and their descriptions for the command-line tool for Help Content Manager. |

Exit Codes

When you run the command-line tool for the Help Content Manager in silent mode, it returns the following exit codes:

```
Success = 0,  
  
FailureToElevate = 100  
InvalidCmdArgs = 101,  
FailOnFetchingOnlineContent = 110,  
FailOnFetchingContentFromDisk = 120,  
FailOnFetchingInstalledBooks = 130,  
NoBooksToUninstall = 200,  
NoBooksToInstall = 300,  
FailOnUninstall = 400,  
FailOnInstall = 500,  
FailOnMove = 600,  
FailOnUpdate = 700,  
FailOnRefresh = 800,  
Cancelled = 900,  
Others = 999,  
ContentManagementDisabled = 1200,  
OnlineHelpPreferenceDisabled = 1201  
UpdateAlreadyRunning = 1300 - (Signals that the update didn't run because another was in progress.)
```

See also

[Help Viewer Administrator Guide](#)

[Help Content Manager Overrides](#)

[Microsoft Help Viewer](#)

Help Content Manager Overrides

12/22/2017 • 1 min to read • [Edit Online](#)

You can change the default behavior of the Help Viewer and Help-related features in the Visual Studio IDE. Some options are specified by creating a `.pkgdef` file to set various registry key values. Others are set directly in the registry.

How to control Help Viewer behavior by using a `.pkgdef` file

1. Create a `.pkgdef` file with the first line as `[$RootKey$\Help]`.
2. Add any or all of the registry key values described in the table below on separate lines, for example
`"UseOnlineHelp"=dword:00000001`.
3. Copy the file to `%ProgramFiles(x86)%\Microsoft Visual Studio\2017\
<edition>\Common7\IDE\CommonExtensions`.
4. Run `devenv /updateconfiguration` in a Developer Command Prompt.

Registry key values

| REGISTRY KEY VALUE | TYPE | DATA | DESCRIPTION |
|------------------------------|--------|--|--|
| NewContentAndUpdateService | string | <http URL for service endpoint> | Define a unique service endpoint |
| UseOnlineHelp | dword | <code>0</code> to specify local Help, <code>1</code> to specify online Help | Define online or offline Help default |
| OnlineBaseUrl | string | <http URL for service endpoint> | Define a unique F1 endpoint |
| OnlineHelpPreferenceDisabled | dword | <code>0</code> to enable or <code>1</code> to disable online Help preference option | Disable online Help preference option |
| DisableManageContent | dword | <code>0</code> to enable or <code>1</code> to disable the Manage Content tab in Help Viewer | Disable the Manage Content tab |
| DisableFirstRunHelpSelection | dword | <code>0</code> to enable or <code>1</code> to disable help features that are configured the first time that Visual Studio starts | Disable installation of content at first launch of Visual Studio |

Example `.pkgdef` file contents

```

[$RootKey$\Help]
"NewContentAndUpdateService"="https://some.service.endpoint"
"UseOnlineHelp"=dword:00000001
"OnlineBaseUrl"="https://some.service.endpoint"
"OnlineHelpPreferenceDisabled"=dword:00000000
"DisableManageContent"=dword:00000000
"DisableFirstRunHelpSelection"=dword:00000001

```

Using Registry Editor to change Help Viewer behavior

The following two behaviors can be controlled by setting registry key values in the Registry Editor.

| TASK | REGISTRY KEY | VALUE | DATA |
|---|---|--------------|---|
| Override BITS job priority | HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node (on a 64-bit machine)\Microsoft\Help\v2.3 | BITSPriority | foreground, high, normal, or low |
| Point to local content store on network share | HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Help\v2.3\Catalogs\VisualStudio15 | LocationPath | "ContentStoreNetworkShare" |

See also

[Help Viewer Administrator Guide](#)

[Command-Line Arguments for the Help Content Manager](#)

[Microsoft Help Viewer](#)

[Modifying the isolated shell by using the .pkgdef file](#)

Visual Studio 2017 workload and component IDs

3/22/2018 • 2 min to read • [Edit Online](#)

Click the edition names in the following table to see the available workload and component IDs you need to install Visual Studio by using a command line, or to specify as a dependency in a VSIX manifest.

| EDITION | ID | DESCRIPTION |
|--------------------------------------|---|--|
| Visual Studio Enterprise 2017 | Microsoft.VisualStudio.Product.Enterprise | Microsoft DevOps solution for productivity and coordination across teams of any size |
| Visual Studio Professional 2017 | Microsoft.VisualStudio.Product.Professional | Professional developer tools and services for small teams |
| Visual Studio Community 2017 | Microsoft.VisualStudio.Product.Community | Free, fully featured IDE for students, open-source, and individual developers |
| Visual Studio Team Explorer 2017 | Microsoft.VisualStudio.Product.TeamExplorer | Interact with Team Foundation Server and Visual Studio Team Services without a Visual Studio developer toolset |
| Visual Studio Desktop Express 2017 | Microsoft.VisualStudio.Workload.WDExpress | Build Native and Managed applications like WPF, WinForms, and Win32 with syntax-aware code editing, source code control, and work item management. Includes support for C#, Visual Basic, and Visual C++. |
| Visual Studio Build Tools 2017 | Microsoft.VisualStudio.Product.BuildTools | The Visual Studio Build Tools allows you to build native and managed MSBuild-based applications without requiring the Visual Studio IDE. There are options to install the Visual C++ compilers and libraries, MFC, ATL, and C++/CLI support. |
| Visual Studio Test Agent 2017 | Microsoft.VisualStudio.Product.TestAgent | Supports running automated tests and load tests remotely |
| Visual Studio Test Controller 2017 | Microsoft.VisualStudio.Product.TestController | Distribute automated tests to multiple machines |
| Visual Studio Test Professional 2017 | Microsoft.VisualStudio.Product.TestProfessional | Visual Studio Test Professional 2017 |
| Visual Studio Feedback Client 2017 | Microsoft.VisualStudio.Product.FeedbackClient | Visual Studio Feedback Client 2017 |

For more information about how to use these lists, see the [Use command-line parameters to install Visual Studio 2017](#) page and the [How to: Migrate extensibility projects to Visual Studio 2017](#) page.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Visual Studio administrator guide for Visual Studio 2017](#)
- [Create an offline installation of Visual Studio 2017](#)
- [Command-line parameter examples for Visual Studio 2017 installation](#)

Visual Studio 2017 build numbers and release dates

3/27/2018 • 3 min to read • [Edit Online](#)

The following table lists the build numbers and release dates for Visual Studio 2017, to date.

| VERSION | CHANNEL | RELEASE DATE | BUILD VERSION |
|--------------------|---------|-------------------|-----------------|
| 15.6.4 | Release | March 22, 2018 | 15.6.27428.2015 |
| 15.7.0 Preview 2.0 | Preview | March 21, 2018 | 15.7.27520.0 |
| 15.6.3 | Release | March 19, 2018 | 15.6.27428.2011 |
| 15.7.0 Preview 1.0 | Preview | March 13, 2018 | 15.7.27512.0 |
| 15.6.2 | Release | March 13, 2018 | 15.6.27428.2005 |
| 15.0.11 | Release | March 13, 2018 | 15.0.26228.29 |
| 15.6.1 | Release | March 8, 2018 | 15.6.27428.2002 |
| 15.6.1 Preview 1.0 | Preview | March 8, 2018 | 15.6.27428.2002 |
| 15.6.0 | Release | March 5, 2018 | 15.6.27428.1 |
| 15.6.0 Preview 7.0 | Preview | March 2, 2018 | 15.6.27428.1 |
| 15.6.0 Preview 6.0 | Preview | February 23, 2018 | 15.6.27421.1 |
| 15.0.10 | Release | February 21, 2018 | 15.0.26228.28 |
| 15.5.7 | Release | February 20, 2018 | 15.5.27130.2036 |
| 15.6.0 Preview 5.0 | Preview | February 14, 2018 | 15.6.27413.0 |
| 15.6.0 Preview 4.0 | Preview | February 7, 2018 | 15.6.27406.0 |
| 15.0.9 | Release | February 2, 2018 | 15.0.26228.23 |
| 15.5.6 | Release | January 29, 2018 | 15.5.27130.2027 |
| 15.5.5 | Release | January 25, 2018 | 15.5.27130.2026 |
| 15.6.0 Preview 3.0 | Preview | January 25, 2018 | 15.6.27323.2 |
| 15.5.4 | Release | January 16, 2018 | 15.5.27130.2024 |
| 15.6.0 Preview 2.0 | Preview | January 10, 2018 | 15.6.27309.0 |

| VERSION | CHANNEL | RELEASE DATE | BUILD VERSION |
|--------------------|---------|--------------------|------------------|
| 15.5.3 | Release | January 9, 2018 | 15.5.27130.2020 |
| 15.0.8 | Release | January 9, 2018 | 15.0.26228.21 |
| 15.5.2 | Release | December 14, 2017 | 15.5.27130.2010 |
| 15.6.0 Preview 1.1 | Preview | December 14, 2017 | 15.6.27205.2004 |
| 15.5.1 | Release | December 7, 2017 | 15.5.27130.2003 |
| 15.6.0 Preview 1.0 | Preview | December 7, 2017 | 15.6.27205.0 |
| 15.0.7 | Release | December 6, 2017 | 15.0.26228.18 |
| 15.5 | Release | December 4, 2017 | 15.5.27130.0 |
| 15.5.0 Preview 5.0 | Preview | November 30, 2017 | 15.5.27128.1 |
| 15.4.5 | Release | November 27, 2017 | 15.4.27004.2010 |
| 15.5.0 Preview 4.0 | Preview | November 14, 2017 | 15.5.27110.0 |
| 15.4.4 | Release | November 14, 2017 | 15.4.27004.2009 |
| 15.0.6 | Release | November 14, 2017 | 15.0.26228.17 |
| 15.4.3 | Release | November 8, 2017 | 15.4.27004.2008 |
| 15.5.0 Preview 3.0 | Preview | November 6, 2017 | 15.5.27102.0 |
| 15.4.2 | Release | October 31, 2017 | 15.4.27004.2006 |
| 15.5.0 Preview 2.0 | Preview | October 23, 2017 | 15.5.27019.1 |
| 15.4.1 | Release | October 19, 2017 | 15.4.27004.2005 |
| 15.5 Preview 1.0 | Preview | October 11, 2017 | 15.0.27009.1 |
| 15.4 | Release | October 9, 2017 | 15.0.27004.20002 |
| 15.4 Preview 6.0 | Preview | October 9, 2017 | 15.0.27004.20002 |
| 15.4 Preview 5.0 | Preview | October 6, 2017 | 15.0.27004.2000 |
| 15.4 Preview 4.0 | Preview | October 2, 2017 | 15.0.26929.2 |
| 15.4 Preview 3.0 | Preview | September 21, 2017 | 15.0.26923.00 |
| 15.3.5 | Release | September 19, 2017 | 15.0.26730.16 |

| VERSION | CHANNEL | RELEASE DATE | BUILD VERSION |
|--------------------|---------|--------------------|---------------|
| 15.0.5 | Release | September 18, 2017 | 15.0.26228.16 |
| 15.3.4 | Release | September 12, 2017 | 15.0.26730.15 |
| 15.4 Preview 2.0 | Preview | September 11, 2017 | 15.0.26906.1 |
| 15.3.3 | Release | August 29, 2017 | 15.0.26730.12 |
| 15.4 Preview 1.0 | Preview | August 24, 2017 | 15.0.26823.01 |
| 15.3.2 | Release | August 22, 2017 | 15.0.26730.10 |
| 15.3.1 | Release | August 18, 2017 | 15.0.26730.08 |
| 15.3.1 Preview 1.0 | Preview | August 18, 2017 | 15.0.26730.08 |
| 15.3.1 | Release | August 18, 2017 | 15.0.26730.08 |
| 15.4 Preview 1.0 | Preview | August 24, 2017 | 15.4.26823.1 |
| 15.3 | Release | August 14, 2017 | 15.0.26730.3 |
| 15.3 Preview 7.1 | Preview | August 11, 2017 | 15.0.26730.3 |
| 15.3 Preview 7.0 | Preview | August 1, 2017 | 15.0.26730.0 |
| 15.3 Preview 6.0 | Preview | July 26, 2017 | 15.0.26724.1 |
| 15.3 Preview 5.0 | Preview | July 24, 2017 | 15.0.26720.02 |
| 15.2.6 | Release | July 17, 2017 | 15.0.26430.16 |
| 15.3 Preview 4.0 | Preview | July 12, 2017 | 15.0.26711.1 |
| 15.2.5 | Release | July 6, 2017 | 15.0.26430.15 |
| 15.3 Preview 3.0 | Preview | June 26, 2017 | 15.0.26621.2 |
| 15.2.4 | Release | June 21, 2017 | 15.0.26430.14 |
| 15.3 Preview 2.1 | Preview | June 20, 2017 | 15.0.26608.5 |
| 15.2.3 | Release | June 9, 2017 | 15.0.26430.13 |
| 15.3 Preview 2.0 | Preview | June 8, 2017 | 15.0.26606.0 |
| 15.2.2 | Release | May 30, 2017 | 15.0.26430.12 |
| 15.0.4 | Release | May 23, 2017 | 15.0.26228.13 |

| Version | Channel | Release Date | Build Version |
|------------------|---------|----------------|---------------|
| 15.2.1 | Release | May 12, 2017 | 15.0.26430.6 |
| 15.3 Preview 1.1 | Preview | May 11, 2017 | 15.0.26510.0 |
| 15.3 Preview 1.0 | Preview | May 10, 2017 | 15.0.26507.0 |
| 15.2 | Release | May 10, 2017 | 15.0.26430.4 |
| 15.2 Preview 4.0 | Preview | May 3, 2017 | 15.0.26430.1 |
| 15.2 Preview 3.0 | Preview | April 26, 2017 | 15.0.26424.2 |
| 15.2 Preview 2.0 | Preview | April 20, 2017 | 15.0.26419.1 |
| 15.2 Preview 1.0 | Preview | April 17, 2017 | 15.0.26412.1 |
| 15.1.2 | Release | April 17, 2017 | 15.0.26403.7 |
| 15.1.1 | Release | April 10, 2017 | 15.0.26403.3 |
| 15.1 | Release | April 5, 2017 | 15.0.26403.0 |
| 15.0.3 | Release | March 31, 2017 | 15.0.26228.12 |
| 15.0.2 | Release | March 28, 2017 | 15.0.26228.10 |
| 15.1 Preview 3.0 | Preview | March 27, 2017 | 15.0.26323.1 |
| 15.1 Preview 2.0 | Preview | March 16, 2017 | 15.0.26315.0 |
| 15.0.1 | Release | March 14, 2017 | 15.0.26228.9 |
| 15.1 Preview 1.0 | Preview | March 7, 2017 | 15.0.26304.0 |
| 15.0.0 | Release | March 7, 2017 | 15.0.26228.4 |

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Visual Studio 2017 administrator guide](#)
- [Use command-line parameters to install Visual Studio 2017](#)
- [Tools for detecting and managing Visual Studio instances](#)

Visual Studio images on Azure

3/7/2018 • 4 min to read • [Edit Online](#)

Using Visual Studio in a preconfigured Azure virtual machine (VM) is a quick, easy way to go from nothing to an up-and-running development environment. System images with different Visual Studio configurations are available in the [Azure Marketplace](#).

New to Azure? [Create a free Azure account](#).

What configurations and versions are available?

Images for the most recent major versions, Visual Studio 2017 and Visual Studio 2015, can be found in the Azure Marketplace. For each major version, you see the originally released (RTW) version and the latest updated versions. Each of these versions offers the Visual Studio Enterprise and the Visual Studio Community editions. These images are updated at least every month to include the latest Visual Studio and Windows updates. While the names of the images remain the same, each image's description includes the installed product version and the image's "as of" date.

| RELEASE VERSION | EDITIONS | PRODUCT VERSION |
|---|-----------------------|-------------------------|
| Visual Studio 2017: Latest (Version 15.6) | Enterprise, Community | Version 15.6.0 |
| Visual Studio 2017: RTW | Enterprise, Community | Version 15.0.10 |
| Visual Studio 2015: Latest (Update 3) | Enterprise, Community | Version 14.0.25431.01 |
| Visual Studio 2015: RTW | None | (Expired for servicing) |

NOTE

In accordance with Microsoft servicing policy, the originally released (RTW) version of Visual Studio 2015 has expired for servicing. Visual Studio 2015 Update 3 is the only remaining version offered for the Visual Studio 2015 product line.

For more information, see the [Visual Studio Servicing Policy](#).

What features are installed?

Each image contains the recommended feature set for that Visual Studio edition. Generally, the installation includes:

- All available workloads, including each workload's recommended optional components
- .NET 4.6.2 and .NET 4.7 SDKs, Targeting Packs, and Developer Tools
- Visual F#
- GitHub Extension for Visual Studio
- LINQ to SQL Tools

We use the following command line to install Visual Studio when building the images:

```
vs_enterprise.exe --allWorkloads --includeRecommended --passive ^
    add Microsoft.Net.Component.4.7.SDK ^
    add Microsoft.Net.Component.4.7.TargetingPack ^
    add Microsoft.Net.Component.4.6.2.SDK ^
    add Microsoft.Net.Component.4.6.2.TargetingPack ^
    add Microsoft.Net.ComponentGroup.4.7.DeveloperTools ^
    add Microsoft.VisualStudio.Component.FSharp ^
    add Component.GitHub.VisualStudio ^
    add Microsoft.VisualStudio.Component.LinqToSql
```

If the images don't include a Visual Studio feature that you require, provide feedback through the feedback tool in the upper-right corner of the page.

What size VM should I choose?

Azure offers a full range of virtual machine sizes. Because Visual Studio is a powerful, multi-threaded application, you want a VM size that includes at least two processors and 7 GB of memory. We recommend the following VM sizes for the Visual Studio images:

- Standard_D2_v3
- Standard_D2s_v3
- Standard_D4_v3
- Standard_D4s_v3
- Standard_D2_v2
- Standard_D2S_v2
- Standard_D3_v2

For more information on the latest machine sizes, see [Sizes for Windows virtual machines in Azure](#).

With Azure, you can rebalance your initial choice by resizing the VM. You can either provision a new VM with a more appropriate size, or resize your existing VM to different underlying hardware. For more information, see [Resize a Windows VM](#).

After the VM is running, what's next?

Visual Studio follows the "bring your own license" model in Azure. As with an installation on proprietary hardware, one of the first steps is licensing your Visual Studio installation. To unlock Visual Studio, either:

- Sign in with a Microsoft account that's associated with a Visual Studio subscription
- Unlock Visual Studio with the product key that came with your initial purchase

For more information, see [Sign in to Visual Studio](#) and [How to unlock Visual Studio](#).

How do I save the development VM for future or team use?

The spectrum of development environments is huge, and there's real cost associated with building out the more complex environments. Regardless of your environment's configuration, you can save, or capture, your configured VM as a "base image" for future use or for other members of your team. Then, when booting a new VM, you provision it from the base image rather than the Azure Marketplace image.

A quick summary: Use the System Preparation tool (Sysprep) and shut down the running VM, and then capture (*Figure 1*) the VM as an image through the UI in the Azure portal. Azure saves the `.vhdx` file that contains the image in the storage account of your choosing. The new image then shows up as an Image resource in your subscription's list of resources.



(Figure 1) Capture an image through the Azure portal's UI.

For more information, see [Create a managed image of a generalized VM in Azure](#).

IMPORTANT

Don't forget to use Sysprep to prepare the VM. If you miss that step, Azure can't provision a VM from the image.

NOTE

You still incur some cost for storage of the images, but that incremental cost can be insignificant compared to the overhead costs to rebuild the VM from scratch for each team member who needs one. For instance, it costs a few dollars to create and store a 127-GB image for a month that's reusable by your entire team. However, these costs are insignificant compared to hours each employee invests to build out and validate a properly configured dev box for their individual use.

Additionally, your development tasks or technologies might need more scale, like varieties of development configurations and multiple machine configurations. You can use Azure DevTest Labs to create *recipes* that automate construction of your "golden image." You can also use DevTest Labs to manage policies for your team's running VMs. [Using Azure DevTest Labs for developers](#) is the best source for more information on DevTest Labs.

Next steps

Now that you know about the preconfigured Visual Studio images, the next step is to create a new VM:

- [Create a VM through the Azure portal](#)
- [Windows Virtual Machines overview](#)

Install Build Tools into a Container

3/21/2018 • 6 min to read • [Edit Online](#)

You can install Visual Studio Build Tools into a Windows container to support continuous integration and continuous delivery (CI/CD) workflows. This article guides you through what Docker configuration changes are required as well as what [workloads and components](#) you can install in a container.

[Containers](#) are a great way to package a consistent build system you can use not only in a CI/CD server environment but for development environments as well. You can, for example, mount your source code into a container to be built by a customized environment while you continue to use Visual Studio or other tools to write your code. If your CI/CD workflow uses the same container image, you can rest assured that your code builds consistently. You can use containers for runtime consistency as well, which is common for micro-services using multiple containers with an orchestration system, but is beyond the scope of this article.

If Visual Studio Build Tools does not have what you require to build your source code, these same steps can be used for other Visual Studio products. Do note, however, that Windows containers do not support an interactive user interface so all commands must be automated.

Overview

Using [Docker](#) you create an image from which you can create containers that build your source code. The example Dockerfile installs the latest Visual Studio Build Tools 2017 and some other helpful programs often used for building source code. You can further modify your own Dockerfile to include other tools and scripts to run tests, publish build output, and more.

If you have already installed Docker for Windows, you can skip to step 3.

Step 1. Enable Hyper-V

Hyper-V is not enabled by default. It must be enabled to start Docker for Windows, since currently only Hyper-V isolation is supported for Windows 10.

- [Enable Hyper-V on Windows 10](#)
- [Enable Hyper-V on Windows Server 2016](#)

NOTE

Virtualization must be enabled on your machine. It is typically enabled by default; however, if Hyper-V install fails, refer to your system documentation for how to enable virtualization.

Step 2. Install Docker for Windows

If using Windows 10, you can download and install [Docker Community Edition for Windows](#). You can use PowerShell to [install Docker Enterprise Edition for Windows Server 2016](#) using Desired State Configuration (DSC), or a [package provider](#) for a simple single installation.

Step 3. Switch to Windows Containers

You can only install Build Tools 2017 on Windows, which requires you [switch to Windows containers](#). Windows containers on Windows 10 support only [Hyper-V isolation](#), while Windows containers on Windows Server 2016

support both Hyper-V and process isolation.

Step 4. Expand maximum container disk size

Visual Studio Build Tools - and to a greater extent, Visual Studio - require lots of disk space for all the tools that get installed. Even though our example Dockerfile disables the package cache, the disk size of container images must be increased to accommodate the space required. Currently on Windows, you can only increase disk size by changing the Docker configuration.

On Windows 10:

1. [Right-click on the Docker for Windows icon](#) in the system tray and click **Settings...**
2. Click on the **Daemon** section.
3. Toggle the **Basic** button to **Advanced**.
4. Add the following JSON array property to increase disk space to 120GB (more than enough for Build Tools with room to grow).

```
{  
  "storage-opts": [  
    "size=120GB"  
  ]  
}
```

This property is added to anything you already have. For example, with these changes applied to the default daemon configuration file, you should now see:

```
{  
  "registry-mirrors": [],  
  "insecure-registries": [],  
  "debug": true,  
  "experimental": true,  
  "storage-opts": [  
    "size=120GB"  
  ]  
}
```

5. Click **Apply**.

On Windows Server 2016:

1. Stop the "docker" service:

```
sc.exe stop docker
```

2. From an elevated command prompt, edit "%ProgramData%\Docker\config\daemon.json" (or whatever you specified to `dockerd --config-file`).
3. Add the following JSON array property to increase disk space to 120GB (more than enough for Build Tools with room to grow).

```
{  
  "storage-opts": [  
    "size=120GB"  
  ]  
}
```

This property is added to anything you already have.

4. Save and close the file.
5. Start the "docker" service:

```
sc.exe start docker
```

Step 5. Create and build the Dockerfile

You must save the following example Dockerfile to a new file on your disk. If the file is named simply "Dockerfile", it is recognized by default.

NOTE

This example Dockerfile only excludes older Windows SDKs that cannot be installed into containers. Older releases cause the build command to fail.

1. Open a command prompt.
2. Create a new directory (recommended):

```
mkdir C:\BuildTools
```

3. Change directories to this new directory:

```
cd C:\BuildTools
```

4. Save the following content to C:\BuildTools\Dockerfile.

```
# Use the latest Windows Server Core image.  
FROM microsoft/windowsservercore  
  
# Download useful tools to C:\Bin.  
ADD https://dist.nuget.org/win-x86-commandline/v4.1.0/nuget.exe C:\\Bin\\nuget.exe  
  
# Download the Build Tools bootstrapper outside of the PATH.  
ADD https://aka.ms/vs/15/release/vs_buildtools.exe C:\\TEMP\\vs_buildtools.exe  
  
# Add C:\Bin to PATH and install Build Tools excluding workloads and components with known issues.  
RUN setx /m PATH "%PATH%;C:\Bin"  
    && C:\\TEMP\\vs_buildtools.exe --quiet --wait --norestart --nocache --installPath C:\\BuildTools --all \  
        --remove Microsoft.VisualStudio.Component.Windows10SDK.10240 \  
        --remove Microsoft.VisualStudio.Component.Windows10SDK.10586 \  
        --remove Microsoft.VisualStudio.Component.Windows10SDK.14393 \  
        --remove Microsoft.VisualStudio.Component.Windows81SDK \  
    || IF "%ERRORLEVEL%"=="3010" EXIT 0  
  
# Start developer command prompt with any other commands specified.  
ENTRYPOINT C:\\BuildTools\\Common7\\Tools\\VsDevCmd.bat &&  
  
# Default to PowerShell if no other command specified.  
CMD ["powershell.exe", "-NoLogo", "-ExecutionPolicy", "Bypass"]
```

5. Run the following command within that directory.

```
docker build -t buildtools2017:latest -m 2GB .
```

This command builds the Dockerfile in the current directory using 2GB of memory. The default 1GB is not sufficient when some workloads are installed; however, you may be able to build with only 1GB of memory depending on your build requirements.

The final image is tagged "buildtools2017:latest" so you can easily run it in a container as "buildtools2017" since the "latest" tag is the default if no tag is specified. If you want to use a specific version of Visual Studio Build Tools 2017 in a more [advanced scenario](#), you might instead tag the container with a specific Visual Studio build number as well as "latest" so containers can use a specific version consistently.

Step 6. Using the built image

Now that you have created an image, you can run it in a container to do both interactive and automated builds. The example uses the Developer Command Prompt, so your PATH and other environment variables are already configured.

1. Open a command prompt.
2. Run the container to start a PowerShell environment with all developer environment variables set:

```
docker run -it buildtools2017
```

To use this image for your CI/CD workflow, you can publish it to your own [Azure Container Registry](#) or other internal [Docker registry](#) so servers only need to pull it.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Advanced Example for Containers](#)
- [Known Issues for Containers](#)
- [Visual Studio Build Tools 2017 workload and component IDs](#)

Advanced Example for Containers

3/21/2018 • 3 min to read • [Edit Online](#)

The sample Dockerfile in [Install Build Tools into a Container](#) always uses the latest microsoft/windowsservercore image and the latest Visual Studio Build Tools 2017 installer. If you publish this image to a [Docker registry](#) for others to pull, this image may be okay for many scenarios. In practice, however, it's more common to be specific about what base image you use, what binaries you download, and which tool versions you install.

The following example Dockerfile uses a specific version tag of the microsoft/windowsservercore image. Using a specific tag for a base image is commonplace and makes it easy to remember that building or rebuilding images always has the same basis.

NOTE

You cannot install Visual Studio into microsoft/windowsservercore:10.0.14393.1593, which has known issues launching the installer in a container. For more information, see [known issues](#).

The example also uses a Build Tools 2017 bootstrapper that installs a specific version built at the same time as the bootstrapper. The product could still be updated via the Release channel, but that is not a practical scenario for containers that you would typically rebuild. If you want to get the URLs for a specific channel, you can download the channel from <https://aka.ms/vs/15/release/channel>, open the JSON file, and examine the bootstrapper URLs. For more information, see [Create a network installation of Visual Studio](#).

```

# Use a specific tagged image. Tags can be changed, though that is unlikely for most images.
# You could also use the immutable tag
@sha256:d841bd78721c74f9b88e2700f5f3c2d66b54cb855b8acb4ab2c627a76a46301d
FROM microsoft/windowsservercore:10.0.14393.1770

# Use PowerShell commands to download, validate hashes, etc.
SHELL ["powershell.exe", "-ExecutionPolicy", "Bypass", "-Command", "$ErrorActionPreference='Stop'; $ProgressPreference='SilentlyContinue'; $VerbosePreference = 'Continue';"]

# Download Build Tools 15.4.27004.2005 and other useful tools.
ENV VS_BUILDTOOLS_URI=https://aka.ms/vs/15/release/6e8971476/vs_buildtools.exe \
    VS_BUILDTOOLS_SHA256=D482171C7F2872B6B9D29B116257C6102DBE6ABA481FAE4983659E7BF67C0F88 \
    NUGET_URI=https://dist.nuget.org/win-x86-commandline/v4.1.0/nuget.exe \
    NUGET_SHA256=4C1DE9B026E0C4AB087302FF75240885742C0FAA62BD2554F913BBE1F6CB63A0

# Download tools to C:\Bin and install Build Tools excluding workloads and components with known issues.
RUN New-Item -Path C:\Bin, C:\TEMP -Type Directory | Out-Null; \
    [System.Environment]::SetEnvironmentVariable('PATH', "\"$env:PATH";C:\Bin\"", 'Machine'); \
    function Fetch ([string] $Uri, [string] $Path, [string] $Hash) { \
        Invoke-RestMethod -Uri $Uri -OutFile $Path; \
        if ($Hash -and ((Get-FileHash -Path $Path -Algorithm SHA256).Hash -ne $Hash)) { \
            throw "\"Download hash for '$Path' incorrect\""; \
        } \
    }; \
    Fetch -Uri $env:NUGET_URI -Path C:\Bin\NuGet.exe -Hash $env:NUGET_SHA256; \
    Fetch -Uri $env:VS_BUILDTOOLS_URI -Path C:\TEMP\vs_buildtools.exe -Hash $env:VS_BUILDTOOLS_SHA256; \
    Fetch -Uri 'https://aka.ms/vscollect.exe' -Path C:\TEMP\collect.exe; \
    $p = Start-Process -Wait -PassThru -FilePath C:\TEMP\vs_buildtools.exe -ArgumentList '--quiet --wait --norestart --nocache --installPath C:\BuildTools --all --remove
Microsoft.VisualStudio.Component.Windows10SDK.10240 --remove
Microsoft.VisualStudio.Component.Windows10SDK.10586 --remove
Microsoft.VisualStudio.Component.Windows10SDK.14393 --remove Microsoft.VisualStudio.Component.Windows81SDK'; \
    if (($ret = $p.ExitCode) -and ($ret -ne 3010)) { C:\TEMP\collect.exe; throw ('Install failed with exit code 0x{0:x}' -f $ret) }

# Restore default shell for Windows containers.
SHELL ["cmd.exe", "/s", "/c"]

# Start developer command prompt with any other commands specified.
ENTRYPOINT C:\BuildTools\Common7\Tools\VsDevCmd.bat &&

# Default to PowerShell if no other command specified.
CMD ["powershell.exe", "-NoLogo", "-ExecutionPolicy", "Bypass"]

```

This example downloads specific tools and validates that the hashes match. It also downloads the latest Visual Studio and .NET log collection utility so that if an installation failure does occur, you can copy the logs to your host machine to analyze the failure.

```

> docker build -t buildtools:15.4.27004.2005 -t buildtools:latest -m 2GB .
Sending build context to Docker daemon
...
Step 4/7 : RUN New-Item -Path C:\Bin, C:\TEMP -Type Directory | Out-Null; ...
--> Running in 4b62b4ce3a3c
Install failed with exit code 0x643
At line:1 char:1
+ throw ('Install failed with exit code 0x{0:x}' -f 1603)
+ ~~~~~
+ CategoryInfo          : OperationStopped: (Install failed with exit code 0x643:String) [],
RuntimeException
+ FullyQualifiedErrorId : Install failed with exit code 0x643

> docker cp 4b62b4ce3a3c:C:\Users\ContainerAdministrator\AppData\Local\TEMP\vslogs.zip "%TEMP%\vslogs.zip"

```

After the last line finishes executing, open "%TEMP%\vslogs.zip" on your machine or submit an issue on the

[Developer Community](#) web site.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Build Tools into a Container](#)
- [Known Issues for Containers](#)
- [Visual Studio Build Tools 2017 workload and component IDs](#)

Known issues for containers

3/21/2018 • 1 min to read • [Edit Online](#)

There are a few issues when installing Visual Studio into a Docker container.

Windows container

The following known issues occur when you install Visual Studio Build Tools 2017 into a Windows container.

- You cannot install Visual Studio into a container based on image `microsoft/windowsservercore:10.0.14393.1593`. Images tagged with Windows versions older or newer should work.
- You cannot install Windows SDK versions older than 10.0.14393. Certain packages fail to install and workloads that depend on those packages will not work.
- You must pass `-m 2GB` (or more) when building the image. Some workloads require more memory than the default 1 GB when installed.
- You must configure Docker to use disks larger than the default 20 GB.
- You must pass `--norestart` on the command line. As of this writing, attempting to restart a Windows container from within the container returns `ERROR_TOO_MANY_OPEN_FILES` to the host.

Build Tools container

The following known issues might occur when you use a Build Tools container. To see whether issues have been fixed or if there are other known issues, visit <https://developercommunity.visualstudio.com>.

- IntelliTrace may not work in [some scenarios](#) within a container.

Get support

Sometimes, things can go wrong. If your Visual Studio installation fails, see the [Troubleshooting Visual Studio 2017 installation and upgrade issues](#) page. If none of the troubleshooting steps help, you can contact us by live chat for installation assistance (English only). For details, see the [Visual Studio support page](#).

Here are a few more support options:

- You can report product issues to us via the [Report a Problem](#) tool that appears both in the Visual Studio Installer and in the Visual Studio IDE.
- You can share a product suggestion with us on [UserVoice](#).
- You can track product issues in the [Visual Studio Developer Community](#), and ask questions and find answers.
- You can also engage with us and other Visual Studio developers through our [Visual Studio conversation in the Gitter community](#). (This option requires a [GitHub](#) account.)

See also

- [Install Build Tools into a Container](#)
- [Advanced Example for Containers](#)
- [Visual Studio Build Tools 2017 workload and component IDs](#)

Quickstart: First look at the Visual Studio IDE

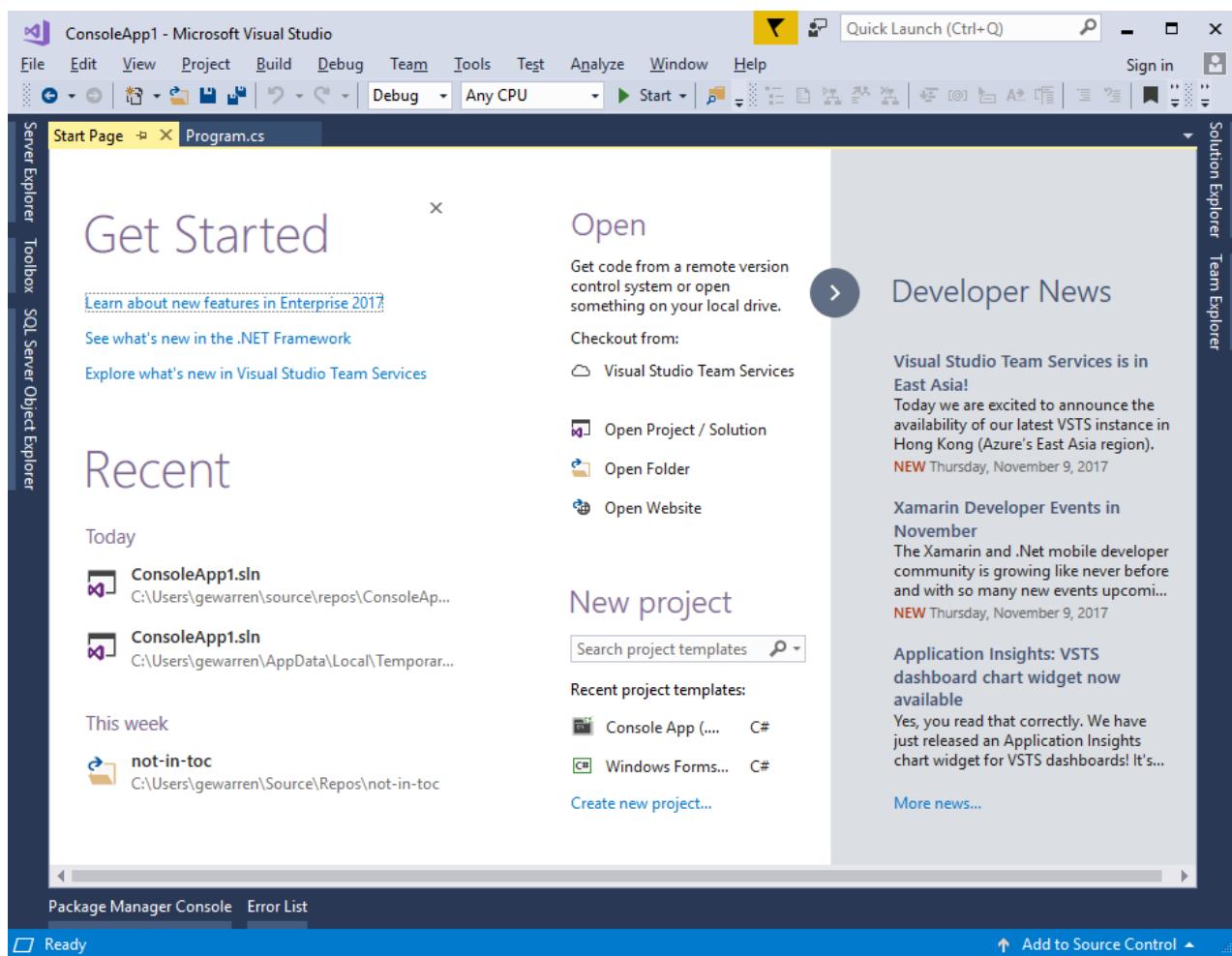
3/12/2018 • 3 min to read • [Edit Online](#)

In this 5-10 minute introduction to the Visual Studio integrated development environment (IDE), we'll take a tour of some of the windows, menus, and other UI features.

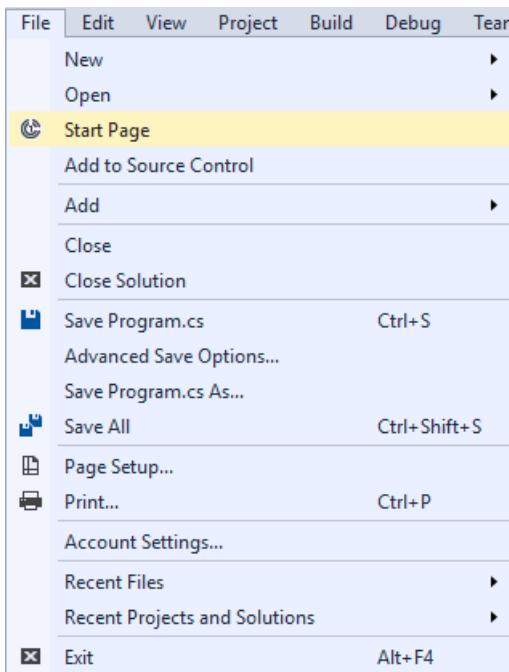
If you haven't already installed Visual Studio, go to the [Visual Studio Downloads](#) page to install it for free.

Start Page

The first thing you'll see after you launch Visual Studio is most likely the Start Page. The Start Page is designed as a "hub" to help you find the commands and project files you need faster. The **Recent** section displays projects and folders you've worked on recently. Under **New project**, you can click a link to bring up the New Project dialog box, or under **Open**, you can open an existing project or code folder. On the right is a feed of the latest developer news.



If you close the Start Page and want to see it again, you can reopen it from the **File** menu.



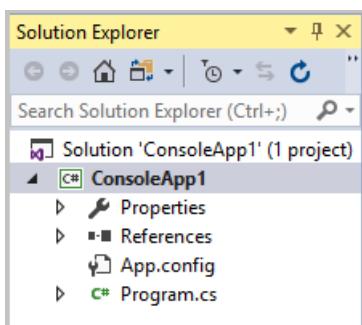
To continue exploring the IDE, let's create a new project.

1. On the **Start Page**, in the search box under **New project**, enter `console` to filter the list of project types. Choose either a C# or VB **Console App (.NET Framework)**. (Alternatively, if you are a C++, Javascript, or other language developer, feel free to create a project in one of those languages.)
2. In the **New Project** dialog box, accept the default project name and choose **OK**.

The project is created and a file named **Program.cs** or **Program.vb** opens in the **Editor** window. The Editor shows the contents of files, and is where you'll do most of your coding work in Visual Studio.

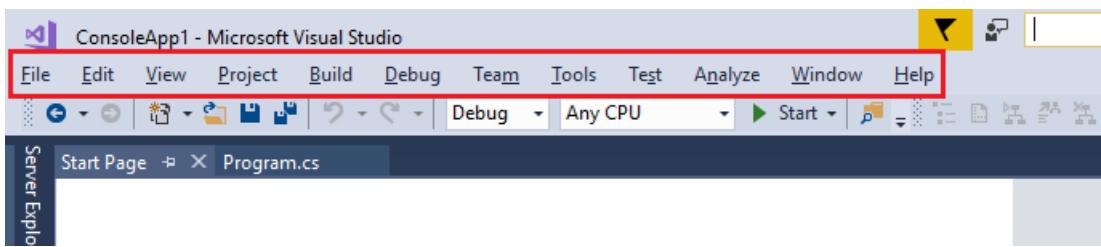
Solution Explorer

Solution Explorer shows you a graphical representation of the hierarchy of files and folders in your project, solution, or code folder. You can browse the hierarchy and navigate to a file in Solution Explorer.



Menus

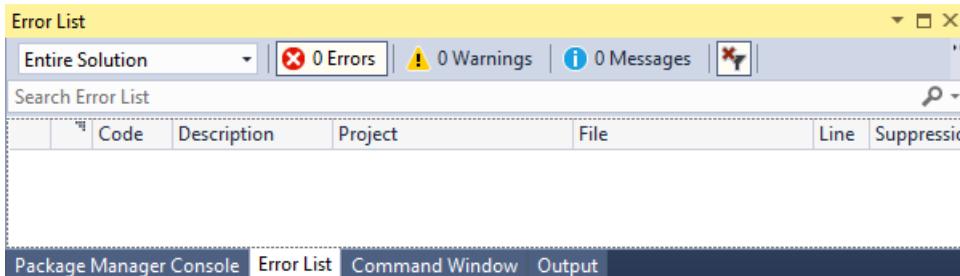
The menu bar along the top of the IDE groups commands into categories. For example, the **Project** menu contains commands related to the project you're working in. On the **Tools** menu, you can customize the IDE by selecting **Options**, or add features to your installation by selecting **Get Tools and Features....**



Let's open the Error List window by choosing the **View** menu, and then **Error List**.

Error List

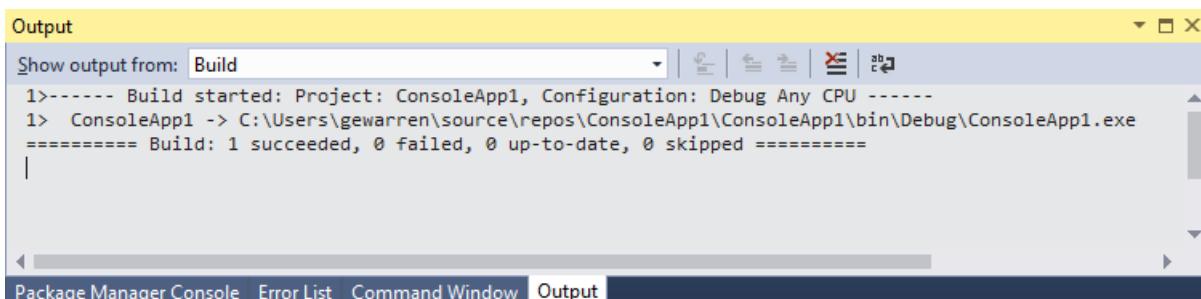
The Error List shows you errors, warning, and messages regarding the current state of your code. If there were any errors (such as a syntax typo) in your file, or anywhere in your project, they would be listed here.



Output window

The Output window shows you output messages from Build and Source Control.

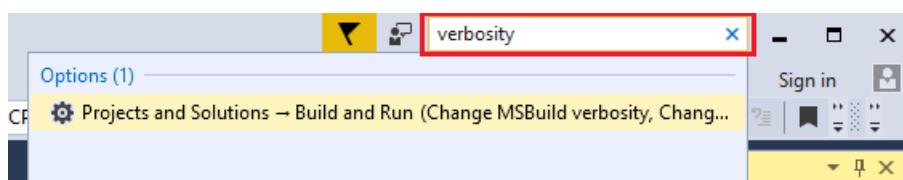
Let's build the project to see some output logging. From the **Build** menu, choose **Build Solution**. The Output window automatically obtains focus and display a successful build message.



Quick Launch

The Quick Launch box is a quick and easy way to do pretty much anything in the IDE. You can enter some text related to what you want to do, and it'll show you a list of options that pertain to the text. For example, let's say we want to increase the build output's verbosity to display additional logging information about what exactly build is doing:

1. Enter `verbosity` into the **Quick Launch** box, and then choose **Projects and Solutions -> Build and Run** under the **Options** category.



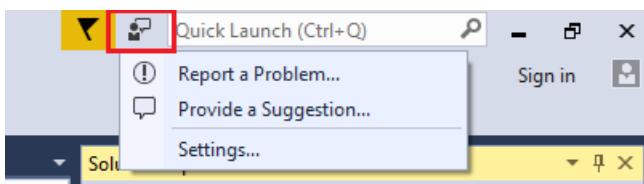
The **Options** dialog box opens to the **Build and Run** options page.

2. Under **MSBuild project build output verbosity**, choose **Normal**, and then click **OK**.
3. Now we'll build the project again by right-clicking on the **ConsoleApp1** project in **Solution Explorer**, and choosing **Rebuild** from the context menu.

This time the Output window shows more verbose logging from the build process, including which files were copied where.

Send Feedback menu

Should you encounter any problems while you're using Visual Studio, or if you have suggestions for how to improve the product, you can use the **Send Feedback** menu at the top of the IDE, next to the Quick Launch box.



Next steps

We've looked at just a few of the features of the Visual Studio IDE to get acquainted with the user interface. To explore further:

- Browse the General User Interface Elements section of the VS documentation, which goes into more depth about windows such as the [Error List](#), [Output window](#), [Properties window](#), and [Options dialog box](#)
- Take a more in-depth tour of the IDE, and even dabble in debugging, in [Overview of the Visual Studio IDE](#)

See also

- [Quickstart: Personalizing the IDE](#)
- [Quickstart: Coding in the editor](#)
- [Quickstart: Projects and solutions](#)

Quickstart: Projects and solutions

3/12/2018 • 6 min to read • [Edit Online](#)

In this 10-minute quickstart, we'll explore what it means to create a solution and a project in Visual Studio. We'll look at the properties of a project and some of the files it can contain. We'll also create a reference to a second project.

If you haven't already installed Visual Studio, go to the [Visual Studio Downloads](#) page to install it for free.

TIP

We'll be constructing a solution and project from scratch in this quickstart, as an educational exercise to understand the concept of a project. In your general use of Visual Studio, you will most likely use the many project templates that Visual Studio offers when you are creating a new project.

NOTE

Solutions and projects are not required to develop apps in Visual Studio. You can also just open a folder that contains code, and start coding, building, and debugging. For example, if you clone a GitHub repo, it might not contain Visual Studio projects and solutions. For more information, see [Develop code in Visual Studio without projects or solutions](#).

Solutions

Solutions are containers used by Visual Studio to organize one or more related projects. When you open a solution in Visual Studio, it will automatically load all the projects it contains.

Create a solution

We'll start our exploration by creating an empty solution. After you get to know Visual Studio, you probably won't find yourself creating empty solutions too often. When you create a new project in Visual Studio, it automatically creates a solution to house the project if there's not a solution already open.

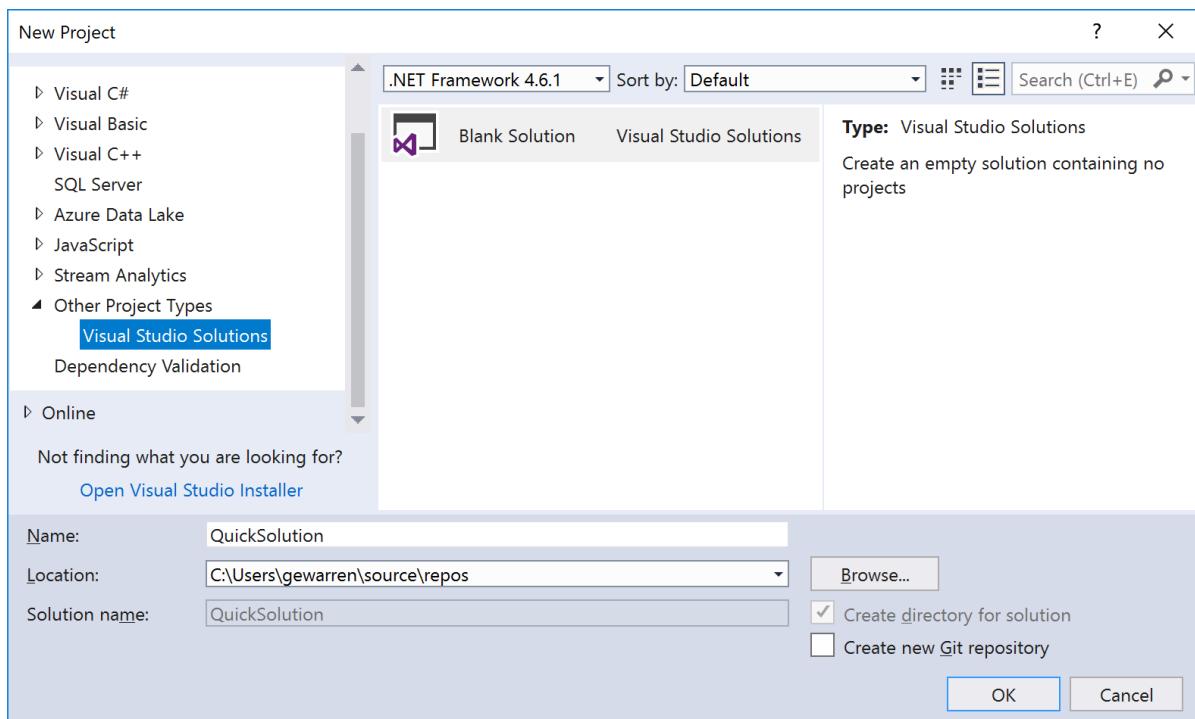
1. Start Visual Studio.

Visual Studio opens, and you'll likely see the **Start Page** taking up most of the window's real estate.

2. On the menu bar, choose **File > New > Project....**

The **New Project** dialog box opens.

3. In the left pane, expand **Other Project Types**, then choose **Visual Studio Solutions**. In the center pane, choose **Blank Solution**. Name your solution "QuickSolution", then choose **OK**.



The **Start Page** closes, and a solution appears in **Solution Explorer** on the right side of the Visual Studio window. You'll probably use **Solution Explorer** often, to browse the contents of your projects.

Add a project

Now let's add our first project to the solution. We'll start with an empty project and add the items we need to the project.

1. From the right-click or context menu of **Solution 'QuickSolution'** in **Solution Explorer**, choose **Add > New Project....**

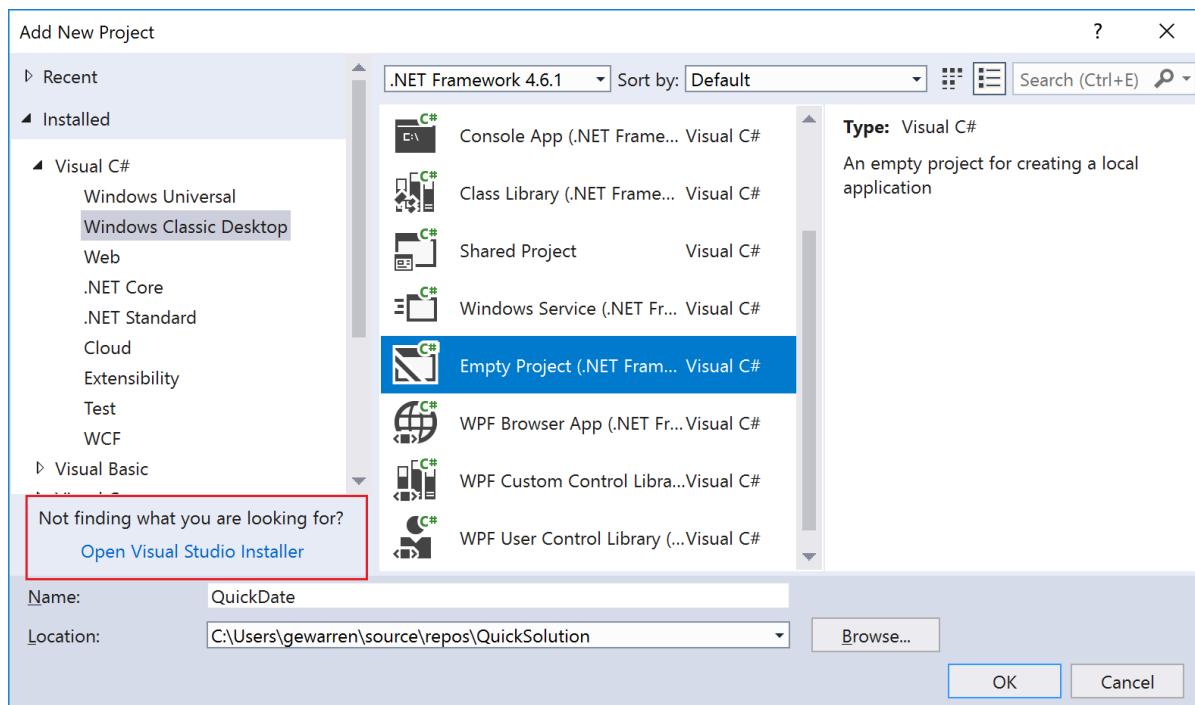
The **Add New Project** dialog box opens.

2. In the left pane, expand **Visual C#** and choose **Windows Classic Desktop**. Then, in the middle pane, choose **Empty Project (.NET Framework)**. Name the project "QuickDate", then choose the **OK** button.

A project named "QuickDate" appears beneath the solution in **Solution Explorer**. Currently it contains a single file called **App.config**.

NOTE

If you don't see **Visual C#** in the left pane of the dialog box, you need to install the **.NET desktop development** workload. An easy way to do this is to choose the **Open Visual Studio Installer** link in the bottom left corner of the dialog box. After **Visual Studio Installer** launches, choose the **.NET desktop development** workload and then the **Modify** button.



Add an item to the project

We have an empty project—let's add a code file.

1. From the right-click or context menu of **QuickDate** in **Solution Explorer**, choose **Add > New Item....**

The **Add New Item** dialog box opens.

2. Expand **Visual C# Items**, then choose **Code**. In the middle pane choose **Class**. Name the class "Calendar", and then choose the **Add** button.

A file named "Calendar.cs" is added to the project. The **.cs** on the end is the file extension that is given to C# code files. The file appears in the visual project hierarchy in **Solution Explorer**, and its contents are opened in the editor.

3. Replace the contents of the **Calendar.cs** file with the following code.

```
using System;

namespace QuickDate
{
    internal class Calendar
    {
        static void Main(string[] args)
        {
            DateTime now = GetCurrentDate();
            Console.WriteLine($"Today's date is {now}");
            Console.ReadLine();
        }

        internal static DateTime GetCurrentDate()
        {
            return DateTime.Now.Date;
        }
    }
}
```

You don't need to understand what the code does, but if you want, you can run the program and see that it prints today's date to the console window.

Add a second project

It is common for solutions to contain more than one project, and often these projects reference each other. Some projects in a solution might be class libraries, some executable applications, and some might be unit test projects or web sites.

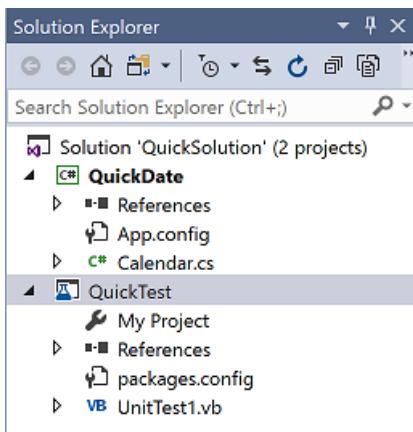
Let's add a unit test project to our solution. This time we'll start from a project template, so we don't have to add an additional code file to the project.

1. From the right-click or context menu of **Solution 'QuickSolution'** in **Solution Explorer**, choose **Add > New Project....**

The **Add New Project** dialog box opens.

2. In the left pane, expand **Visual Basic** and choose the **Test** category. In the middle pane, choose **Unit Test Project (.NET Framework)**. Name the project "QuickTest", and then choose the **OK** button.

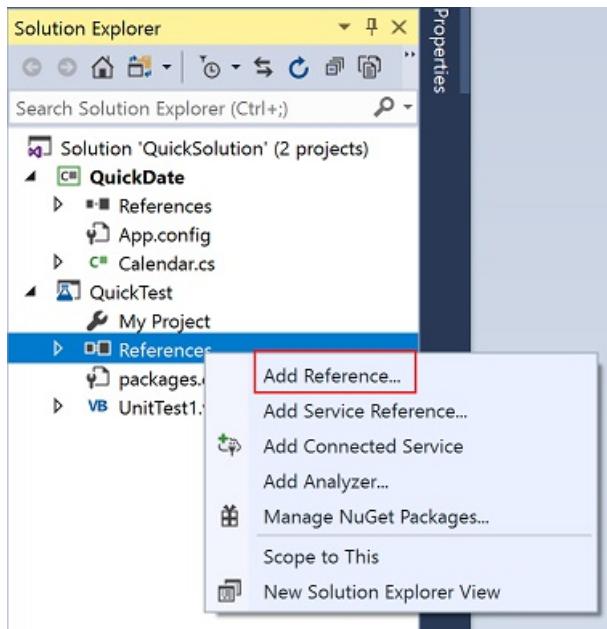
A second project is added to **Solution Explorer**, and a file named **UnitTest1.vb** opens in the editor. **.vb** is the file extension that is given to Visual Basic code files.



Add a project reference

We're going to use the new unit test project to test our method in the **QuickDate** project, so we need to add a reference to that project. This creates a build dependency between the two projects, meaning **QuickDate** will be built before **QuickTest** when the solution is built.

1. Choose the **References** node in the **QuickTest** project, and from the right-click or context menu, choose **Add Reference....**



The **Reference Manager** dialog box opens.

2. In the left pane, expand **Projects** and choose **Solution**. In the middle pane, choose the checkbox next to **QuickDate**, and then choose the **OK** button.

A reference to the **QuickDate** project is added.

Add test code

1. Now we'll add test code to the Visual Basic code file. Replace the contents of **UnitTest1.vb** with the following code.

```
<TestClass()> Public Class UnitTest1

    <TestMethod()> Public Sub TestGetCurrentDate()
        Assert.AreEqual(DateTime.Now.Date, QuickDate.Calendar.GetCurrentDate())
    End Sub

End Class
```

You'll see a red "squiggly" under some of the code. We'll fix this error by making the test project a [friend assembly](#) to the **QuickDate** project.

2. Back in the **QuickDate** project, open the **Calendar.cs** file if it's not already open, and add the following using statement and [InternalsVisibleToAttribute](#) attribute, to resolve the error in the test project.

```
using System.Runtime.CompilerServices;

[assembly: InternalsVisibleTo("QuickTest")]
```

The code file should look like this.

```

1  using System;
2  using System.Runtime.CompilerServices;
3
4  [assembly: InternalsVisibleTo("QuickTest")]
5
6  namespace QuickDate
7  {
8      internal class Calendar
9      {
10         static void Main(string[] args)
11         {
12             DateTime now = GetCurrentDate();
13             Console.WriteLine($"Today's date is {now}");
14             Console.ReadLine();
15         }
16
17         internal static DateTime GetCurrentDate()
18         {
19             return DateTime.Now.Date;
20         }
21     }
22 }
23

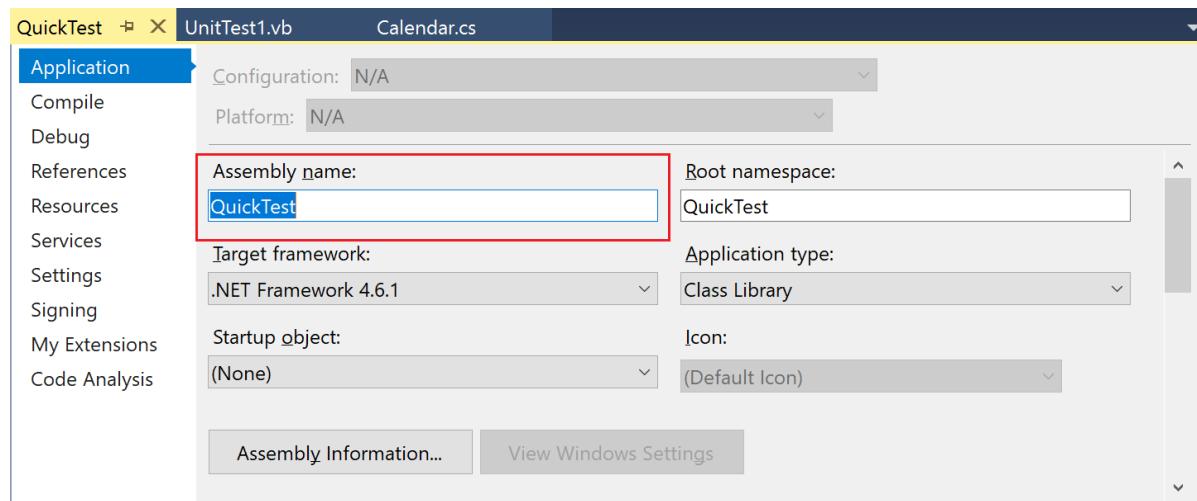
```

Project properties

The line in the C# code file that contains the [InternalsVisibleToAttribute](#) attribute references the assembly name of the **QuickTest** project. The assembly name might not always be the same as the project name. To find the assembly name of a project, open the project properties.

1. In **Solution Explorer**, select the **QuickTest** project. From the right-click or context menu, select **Properties**, or just press **Alt+Enter**.

The property pages for the project open on the **Application** tab. Notice that the assembly name of the **QuickTest** project is indeed "QuickTest". If you wanted to change it, this is where you would change it. Then, when you build the test project, the name of the resulting executable file would change from **QuickTest.exe** to whatever you chose.



2. Explore some of the other tabs of the project's property pages, such as **Compile** and **Settings**. These tabs will be different depending on the type of project.

Next steps

If you want to check that your unit test is working, choose **Test > Run > All Tests** from the menu bar. A window called **Test Explorer** opens, and you should see that the **TestGetCurrentDate** test passes.

Congratulations on completing this quickstart! Next, you might want to explore some of the other quickstarts for Visual Studio, or learn more about [creating projects and solutions](#).

See also

- [Quickstart: First look at the Visual Studio IDE](#)
- [Quickstart: Personalize the Visual Studio IDE and editor](#)
- [Quickstart: Coding in the editor](#)
- [Managing project and solution properties](#)
- [Managing references in a project](#)
- [Develop code in Visual Studio without projects or solutions](#)
- [Visual Studio IDE overview](#)

Quickstart: Use the code editor

3/8/2018 • 5 min to read • [Edit Online](#)

In this 10-minute introduction to the editor, we'll add code to a file to look at some of the ways that Visual Studio makes writing, navigating, and understanding code easier.

If you haven't already installed Visual Studio, go to the [Visual Studio Downloads](#) page to install it for free.

This quickstart assumes you are already familiar with a programming language. If you aren't, we suggest you look at one of the programming quickstarts first, such as create a web app with [Python](#) or [C#](#), or create a console app with [Visual Basic](#) or [C++](#).

Create a new code file

Start by creating a new file and adding some code to it. Notice that we do not have to create a project to gain some of the benefits that the editor offers.

1. Open Visual Studio, and from the **File** menu on the menu bar, choose **New > File....**
2. In the **New File** dialog box, under the **General** category, choose **Visual C# Class**, and then choose **Open**.

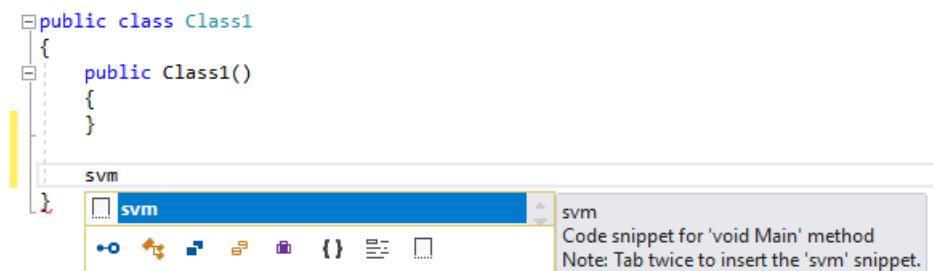
A new file opens in the editor with the skeleton of a C# class.

Using code snippets

Visual Studio provides useful code snippets that you can use to quickly and easily generate commonly used code blocks. [Code snippets](#) are available for different programming languages including C#, Visual Basic, and C++. Let's add the C# `void Main` snippet to our file.

1. Place your cursor below the closing brace of the `Class1` constructor and enter the characters `svm`.

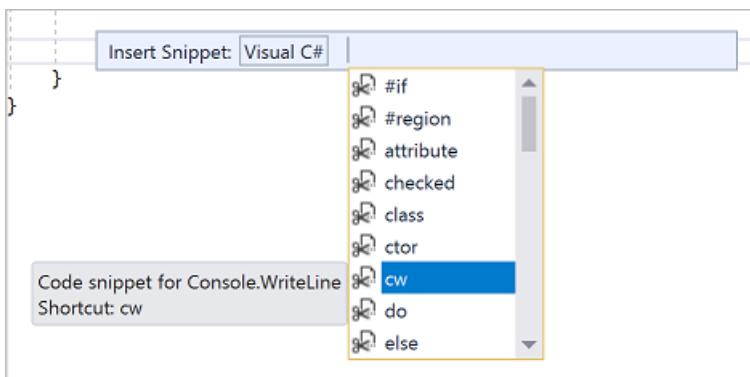
You see an IntelliSense dialog box appear with information about the `svm` snippet.



2. Press **Tab** twice to insert the code snippet.

You see the `static void Main()` method signature get added to the file. The `Main()` method is the entry point for C# applications.

The available code snippets vary for different languages. You can look at the available code snippets for your programming language by choosing **Edit, IntelliSense, Insert Snippet...**, and then choosing your language's folder. For C#, the list looks like this:



The list includes snippets for creating a class, a constructor, `Console.WriteLine()`, `for` loops, `if` and `switch` statements, and more.

Commenting out code

The toolbar provides a number of buttons to make you more productive as you code. For example, you can toggle IntelliSense completion mode, increase or decrease an indent, set a bookmark, or comment out code. In this section, we'll comment out some code that we don't want to compile.



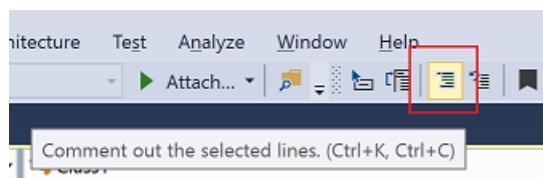
1. Paste the following code into the `Main()` method body.

```
// _words is a string array that we'll sort alphabetically
string[] _words = {
    "the",
    "quick",
    "brown",
    "fox",
    "jumps"
};

string[] morewords = {
    "over",
    "the",
    "lazy",
    "dog"
};

IQueryable<string> query = from word in _words
                           orderby word.Length
                           select word;
```

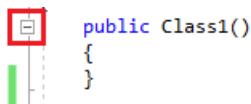
2. We are not using the `morewords` variable, but we may use it later so we don't want to delete it. Instead, let's comment out those lines. Select the entire definition of `morewords` to the closing semi-colon, and then choose the **Comment out the selected lines** button on the toolbar, or press **Ctrl+K, Ctrl+C**.



The C# comment characters `//` are added to the beginning of each selected line to comment out the code.

Collapsing code blocks

We don't want to see the empty constructor for `Class1` that was generated, so to unclutter our view of the code, let's collapse it. Choose the small gray box with the minus sign inside it in the margin of the first line of the constructor. Or, if you are a keyboard user, place the cursor anywhere in the constructor code and press **Ctrl+M**, **Ctrl+M**.



```
public Class1()
{
}
```

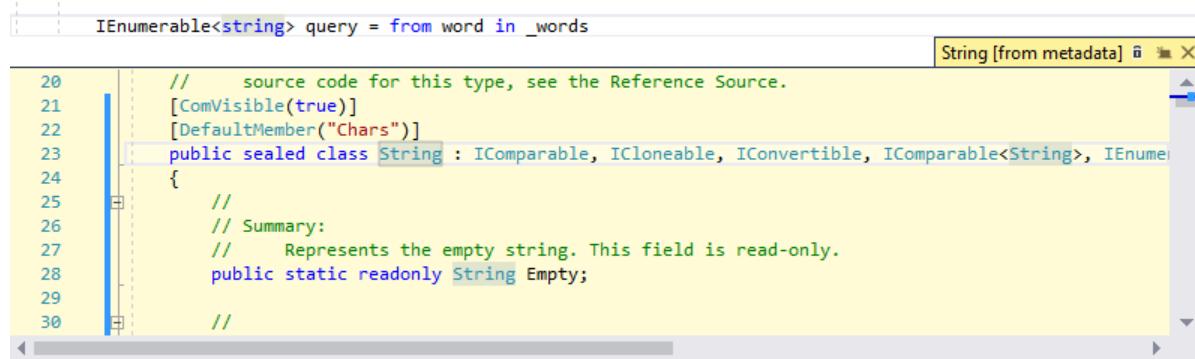
The code block collapses to just the first line, followed by an ellipsis (`...`). To expand the code block again, click the same gray box that now has a plus sign in it, or press **Ctrl+M**, **Ctrl+M** again. This feature is called [outlining](#) and is especially useful when you're collapsing long methods or entire classes.

Viewing symbol definitions

The Visual Studio editor makes it easy to inspect the definition of a type, method, etc. One way is to navigate to the file that contains the definition, for example by choosing **Go to Definition** anywhere the symbol is referenced. An even quicker way that doesn't move your focus away from the file you're working in is to use [Peek Definition](#). Let's peek at the definition of `string`.

1. Right-click on any occurrence of `string` and choose **Peek Definition** from the context menu—or, press **Alt+F12**.

A pop-up window appears with the definition of the `string` class. You can scroll within the pop-up window, or even peek at the definition of another type from the peeked code.



2. Close the peeked definition window by choosing the small box with an "x" at the top right of the pop-up window.

Using IntelliSense to complete words

[IntelliSense](#) is an invaluable resource when you're coding. It can show you information about available members of a type, or parameter details for different overloads of a method. You can also use IntelliSense to complete a word after you type enough characters to disambiguate it. Let's add a line of code to print out the ordered strings to the console window.

1. Below the `query` variable, start typing the following code:

```
foreach (string str in qu
```

You see IntelliSense show you **Quick Info** about the `query` symbol.



2. To insert the rest of the word `query` by using IntelliSense's "Complete Word" functionality, press **Tab**.
3. Finish off the code block to look like the following code. You can even practice using code snippets again by entering `cw` and then pressing **Tab** twice to generate the `Console.WriteLine` code.

```
foreach (string str in query)
{
    Console.WriteLine(str);
}
```

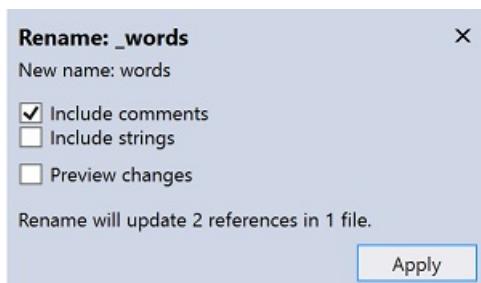
Refactoring a name

Nobody gets code right the first time, and one of the things you might want to change is the name of a variable or method. Let's try out Visual Studio's **refactoring** functionality to rename the `_words` variable to `words`.

1. Place your cursor over the definition of the `_words` variable, and choose **Rename...** from the right-click or context menu, or press **Ctrl+R, Ctrl+R**.

A pop-up **Rename** dialog box appears at the top right of the editor.

2. Enter the desired name `words`. Notice that the reference to `_words` in the query is also automatically renamed. Before you press **Enter**, select the **Include comments** checkbox in the **Rename** pop-up box.



3. Press **Enter**.

Both occurrences of `_words` have been renamed, as well as the reference to `_words` in the code comment.

Next steps

You've completed this quickstart for the Visual Studio editor! Next you might try out some of the other quickstarts for the Visual Studio IDE, look at more ways of [navigating code](#), or check out the links to more information about the features we looked at. Otherwise, happy coding!

See also

- [Quickstart: First look at the Visual Studio IDE](#)
- [Quickstart: Personalize the Visual Studio IDE and editor](#)
- [Quickstart: Projects and solutions](#)
- [Code snippets](#)
- [Outlining](#)
- [Go To Definition and Peek Definition](#)
- [Refactoring](#)

- Using IntelliSense

Quickstart: Personalize the Visual Studio IDE and Editor

3/12/2018 • 2 min to read • [Edit Online](#)

In this 5-10 minute quickstart, we'll customize the Visual Studio color theme and two text colors in the Text Editor.

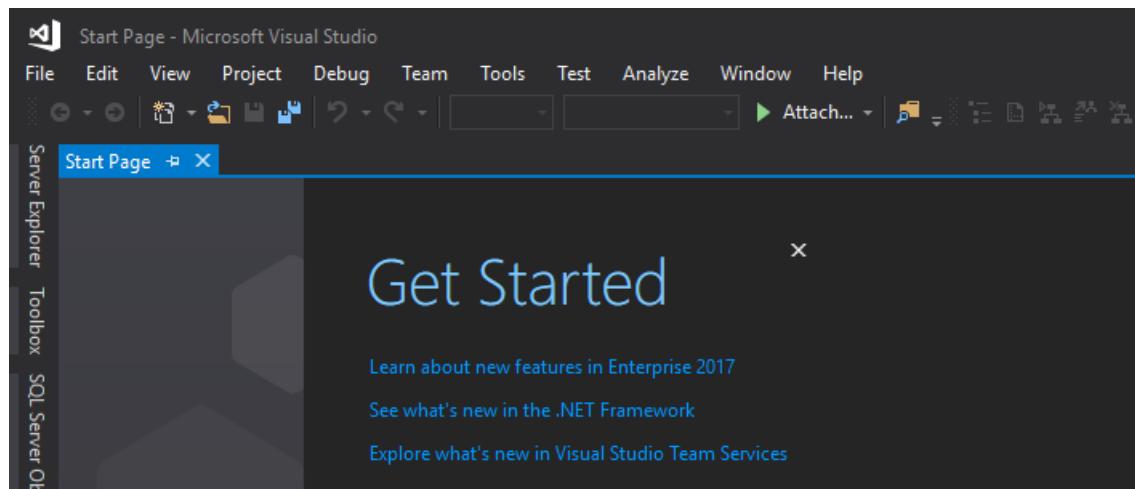
If you haven't already installed Visual Studio, go to the [Visual Studio Downloads](#) page to install it for free.

Set the color theme

The default color theme for Visual Studio 2017 is called **Blue**. Let's change it to **Dark**.

1. On the menu bar, choose **Tools** > **Options**.
2. On the **Environment** > **General** options page, change the **Color theme** selection to **Dark**, and then choose **OK**.

The color theme for the entire IDE is changed to **Dark**.



TIP

You can install additional predefined themes by installing the [Visual Studio Color Theme Editor](#) from the [Visual Studio Marketplace](#). After you install this tool, additional color themes appear in the Color theme drop-down list.

Change text color

Now we'll customize some text colors for the Editor. First, let's open an XML file to see the default colors.

1. From the menu bar, choose **File** > **New** > **File....**
2. In the **New File** dialog box, under the **General** category, choose **XML File**, and then choose **Open**.
3. Paste the following XML below the line that contains `<?xml version="1.0" encoding="utf-8"?>`.

```

<Catalog>
  <Book id="bk101">
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Genre>Computer</Genre>
    <Price>44.95</Price>
    <PublishDate>2000-10-01</PublishDate>
    <Description>
      An in-depth look at creating applications with XML.
    </Description>
  </Book>
  <Book id="bk102">
    <Author>Garcia, Debra</Author>
    <Title>Midnight Rain</Title>
    <Genre>Fantasy</Genre>
    <Price>5.95</Price>
    <PublishDate>2000-12-16</PublishDate>
    <Description>
      A former architect battles corporate zombies, an evil
      sorceress, and her own childhood to become queen of the world.
    </Description>
  </Book>
</Catalog>

```

Notice that the line numbers are a turquoise-blue color, and the xml attributes are a light blue color. We are going to change the text color for these items.

```

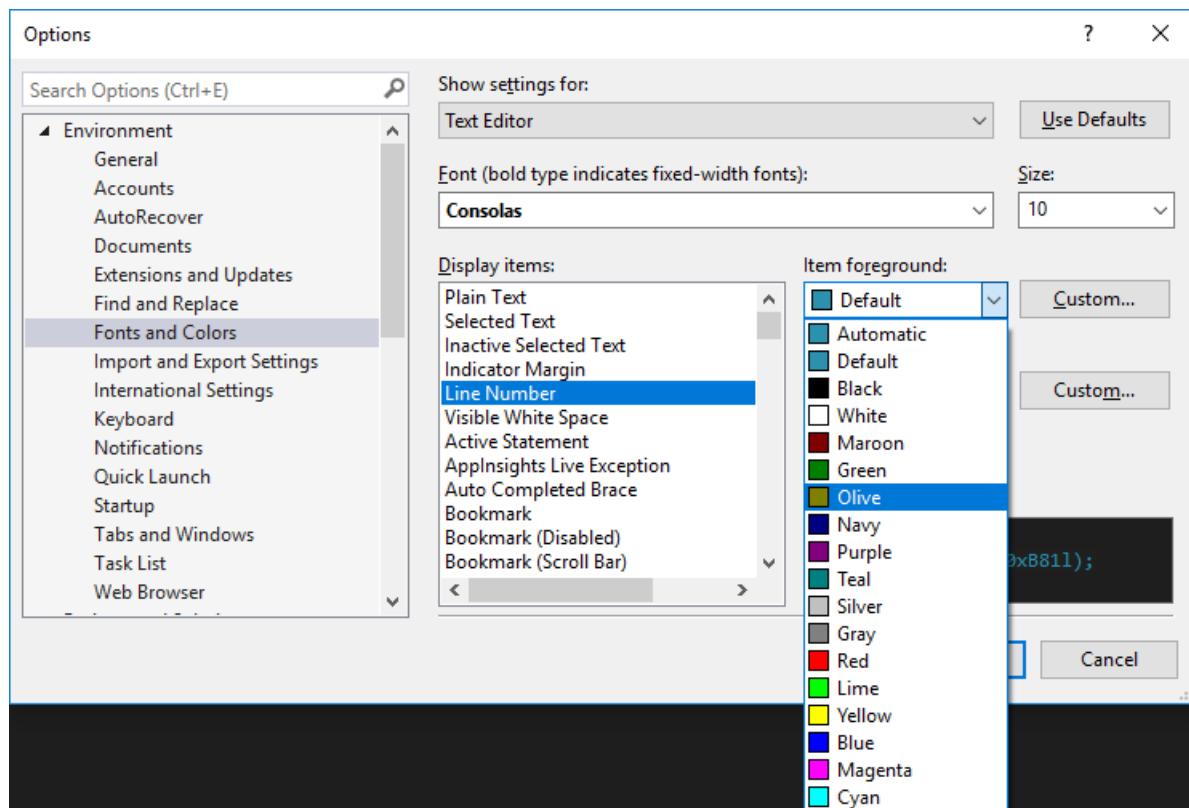
XMLFile1.xml* ✘ X
1  <?xml version="1.0" encoding="utf-8"?>
2  <Catalog>
3    <Book id="bk101">
4      <Author>Garghentini, Davide</Author>
5      <Title>XML Developer's Guide</Title>
6      <Genre>Computer</Genre>
7      <Price>44.95</Price>
8      <PublishDate>2000-10-01</PublishDate>
9      <Description>
10        An in-depth look at creating applications with XML.
11      </Description>
12    </Book>
13    <Book id="bk102">
14      <Author>Garcia, Debra</Author>
15      <Title>Midnight Rain</Title>
16      <Genre>Fantasy</Genre>
17      <Price>5.95</Price>
18      <PublishDate>2000-12-16</PublishDate>
19      <Description>
20        A former architect battles corporate zombies, an evil
21        sorceress, and her own childhood to become queen of the world.
22      </Description>
23    </Book>
24  </Catalog>

```

4. To open the **Options** dialog box, choose **Tools > Options** from the menu bar.
5. Under **Environment**, choose the **FONT AND COLORS** category.

Notice that the text under **Show settings for** says **Text Editor**—this is what we want. You might expand the drop-down list just to see the extensive list of places where you can customize fonts and text color.

6. To change the color of the line numbers text, in the **DISPLAY ITEMS** list, choose **LINE NUMBER**. In the **ITEM FOREGROUND** box, choose **Olive**.



Some languages have their own specific fonts and colors settings. If you are a C++ developer and you want to change the color used for functions, for example, you can look for **C++ Functions** in the **Display items** list.

7. Before we exit out of the dialog box, let's also change the color of XML attributes. In the **Display items** list, scroll down to **XML Attribute** and select it. In the **Item foreground** box, choose **Lime**. Choose **OK** to save our selections and close the dialog box.

The line numbers are now an olive color, and the XML attributes are a bright, lime green. If you open another file type, such as a C++ or C# code file, you'll see that the line numbers also appear in the olive color.

```
<?xml version="1.0" encoding="utf-8"?>
<Catalog>
  <Book id="bk101">
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Genre>Computer</Genre>
    <Price>44.95</Price>
    <PublishDate>2000-10-01</PublishDate>
    <Description>
      An in-depth look at creating applications with XML.
    </Description>
  </Book>
  <Book id="bk102">
    <Author>Garcia, Debra</Author>
    <Title>Midnight Rain</Title>
    <Genre>Fantasy</Genre>
    <Price>5.95</Price>
    <PublishDate>2000-12-16</PublishDate>
    <Description>
      A former architect battles corporate zombies, an evil
      sorceress, and her own childhood to become queen of the world.
    </Description>
  </Book>
</Catalog>
```

We explored just a couple ways of customizing the colors in Visual Studio. We hope that you'll explore the other customization options in the **Options** dialog box, to truly make Visual Studio your own.

See also

- [Quickstart: First look at the Visual Studio IDE](#)
- [Quickstart: Coding in the editor](#)
- [Quickstart: Projects and solutions](#)
- [Personalize the Visual Studio IDE](#)
- [Customizing the Editor](#)
- [Visual Studio IDE Overview](#)

Get Started with C++ in Visual Studio

1/26/2018 • 5 min to read • [Edit Online](#)

Complete this quickstart to become familiar with many of the tools and dialog boxes that you can use when you develop applications in C++ with Visual Studio. Create a "Hello, World"-style console application while you learn more about working in the integrated development environment (IDE).

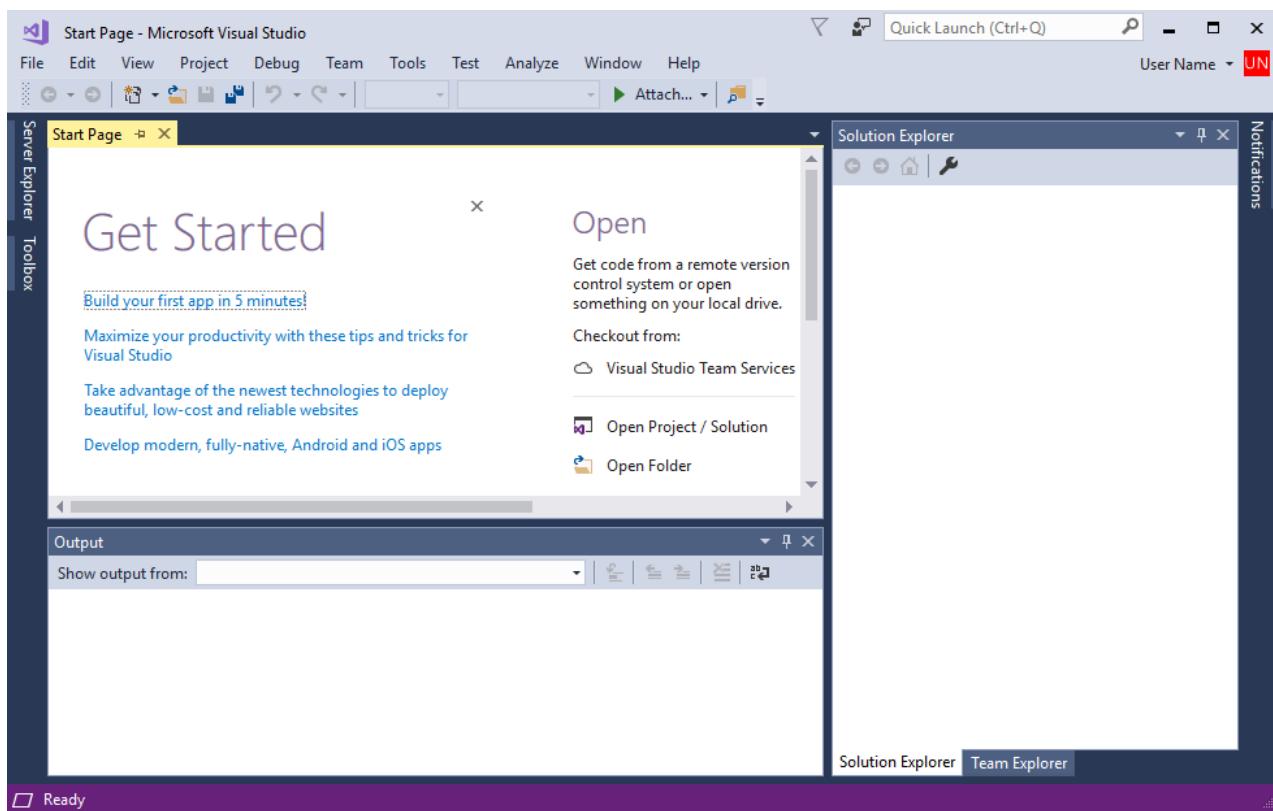
Prerequisites

You don't need to be familiar with C++ to complete this quickstart, but you should be familiar with some general programming and debugging concepts. The Visual Studio documentation doesn't teach you how to program in C++. A good guide to C++ learning resources is the [Get Started](#) page on the ISO C++ website.

To follow along, you need a copy of Visual Studio 2017 version 15.3 or later, with the **Desktop development with C++** workload installed. For a fast guide to installation, see [Install C++ support in Visual Studio](#).

Create a console app

If it's not running yet, start Visual Studio.

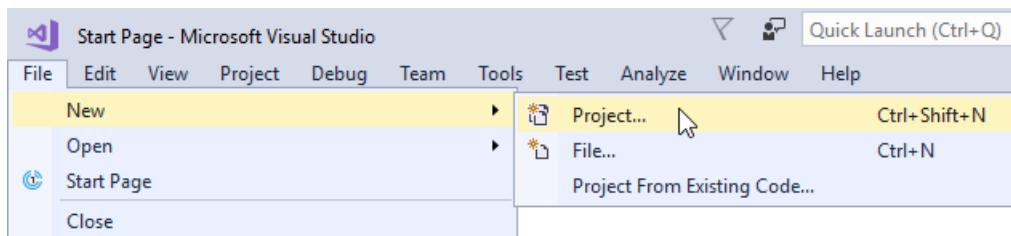


After you open Visual Studio, you can see the three basic parts of the IDE: tool windows, menus and toolbars, and the main window space. Tool windows are docked on the left and right sides of the app window. The **Quick Launch** box, the menu bar, and the standard toolbar are found at the top. The center of the window contains the **Start Page**. When you open a solution or project, editors and designers appear in this space. When you develop an app, most of your time is spent in this central area.

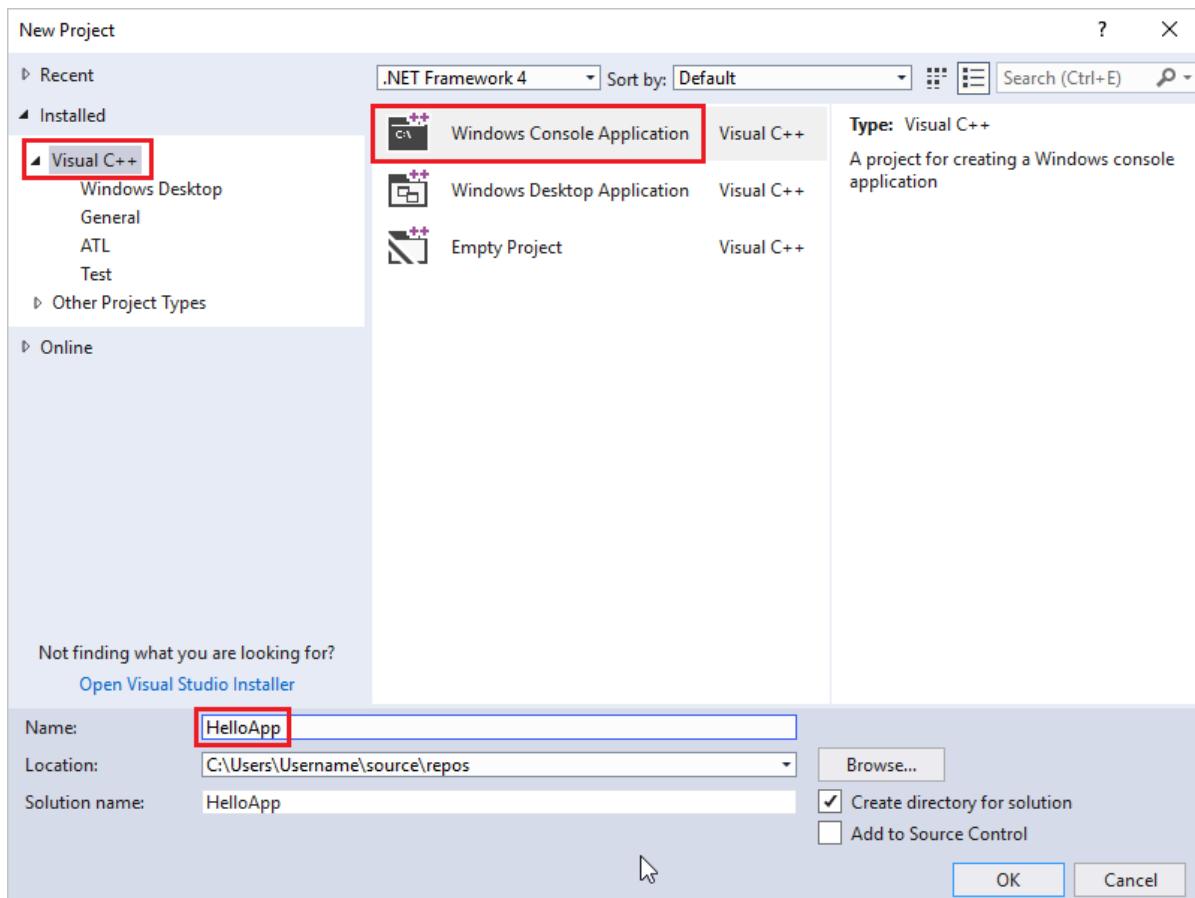
Visual Studio uses *projects* to organize the code for an app, and *solutions* to organize your projects. A project contains all the options, configurations, and rules used to build your apps. It also manages the relationship between all the project's files and any external files. To create your app, first, you create a new project and solution.

To create a console app project

1. On the menu bar, choose **File > New > Project** to open the **New Project** dialog box.



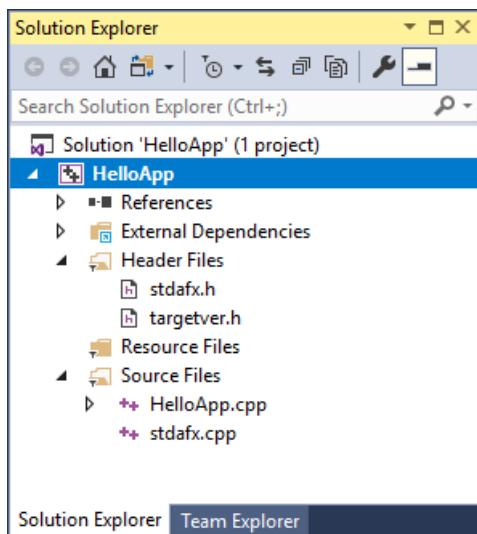
2. In the **New Project** dialog, select **Installed > Visual C++** if it isn't selected already. In the center pane, select the **Windows Console Application** template. In the **Name** edit box, enter *HelloApp*.



Your dialog box may have different choices, depending on the Visual Studio workloads and components you've installed. If you don't see Visual C++ project templates, you need to run the Visual Studio installer again and install the **Desktop development with C++** workload. You can do this directly from the **New Project** dialog. To launch the installer, choose the **Open Visual Studio Installer** link on the dialog.

3. Choose the **OK** button to create your app project and solution.

The HelloApp project and solution, with the basic files for a Windows console app, are created and automatically loaded into **Solution Explorer**. The HelloApp.cpp file is opened in the code editor. These items appear in **Solution Explorer**:



Add code to the app

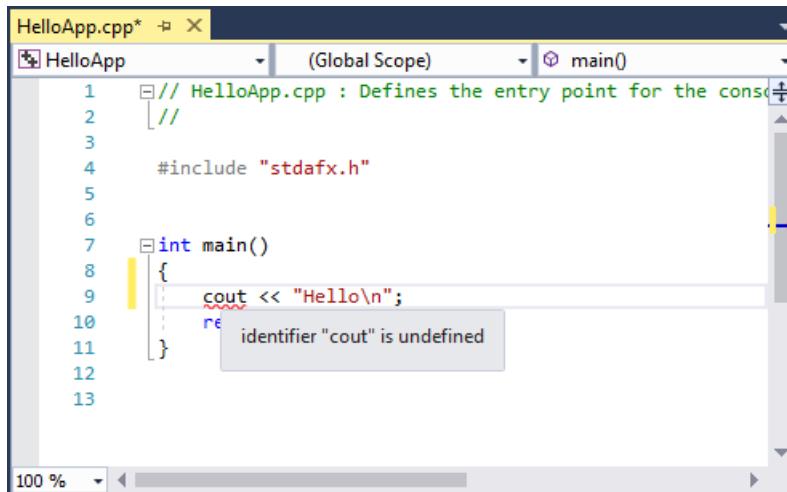
Next, add code to display the word "Hello" in the console window.

To edit code in the editor

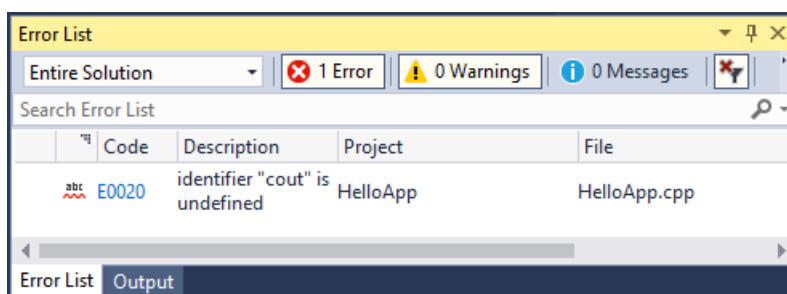
1. In the HelloApp.cpp file, enter a blank line before the line `return 0;` and then enter this code:

```
cout << "Hello\n";
```

A red squiggly line appears under `cout`. If you hover the pointer over it, an error message appears.



The error message also appears in the **Error List** window. You can display this window by choosing **View > Error List** on the menu bar.

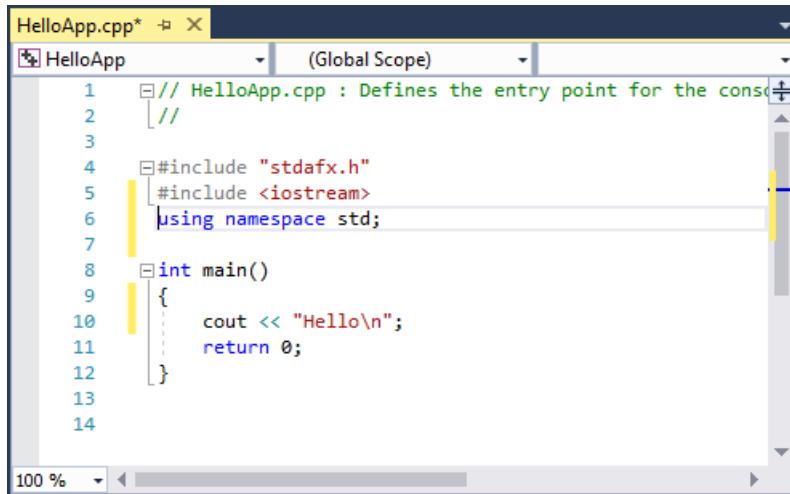


Your code is missing a declaration for `std::cout`, which is found in the `<iostream>` header file.

2. To include the iostream header, enter this code after `#include "stdafx.h"`:

```
#include <iostream>
using namespace std;
```

You probably noticed that a box appeared as you entered code. This box contains auto-completion suggestions for the characters that you enter. It's part of C++ IntelliSense, which provides coding prompts, including class or interface members and parameter information. You can also use code snippets, which are pre-defined blocks of code. For more information, see [Using IntelliSense](#) and [Code Snippets](#).

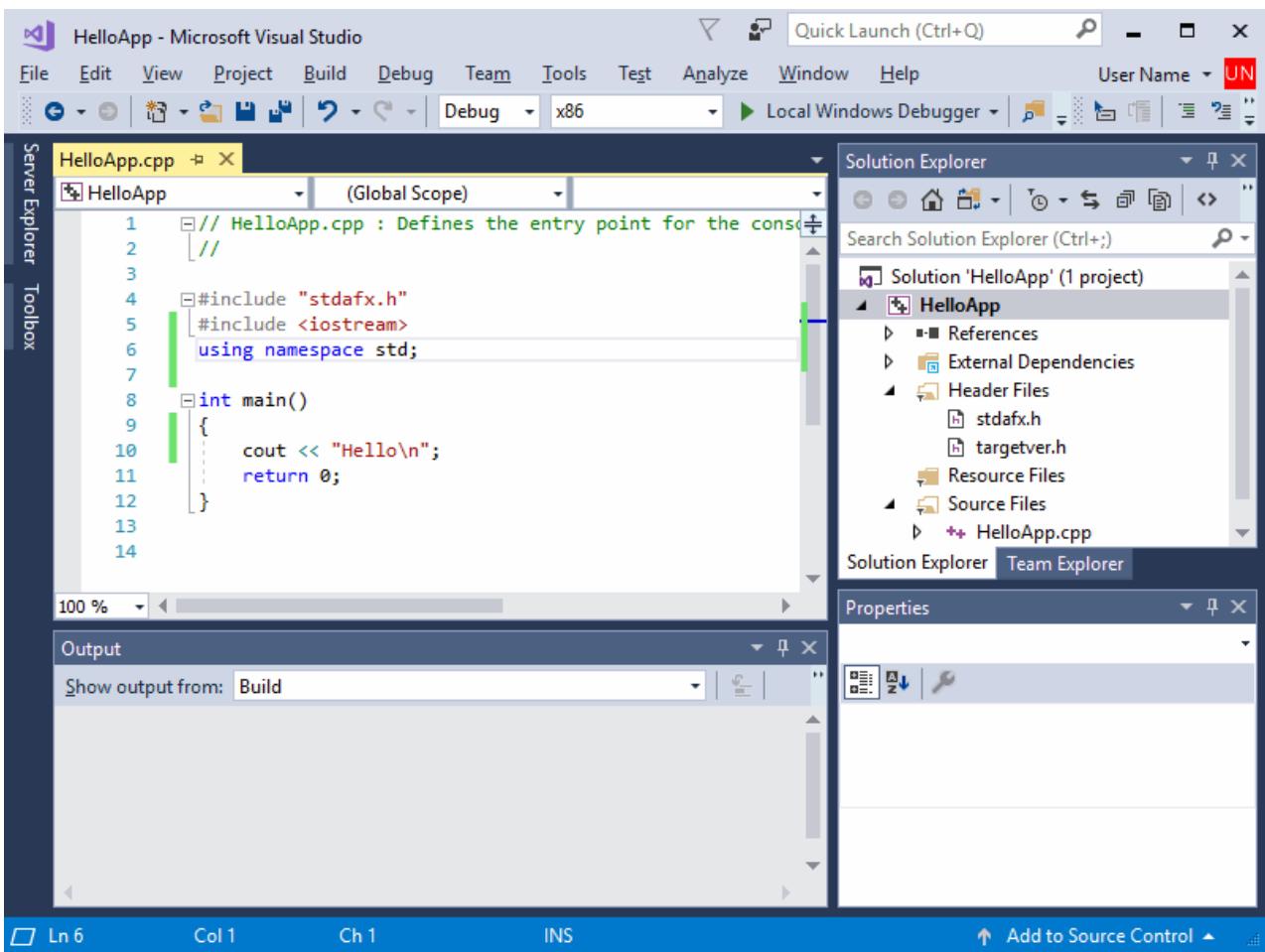


The red squiggly line under `cout` disappears when you fix the error.

3. To save the changes to the file, press **Ctrl+S**.

Build the app

It's easy to build your code. On the menu bar, choose **Build > Build Solution**. Visual Studio builds the HelloApp solution, and reports progress in the **Output** window.

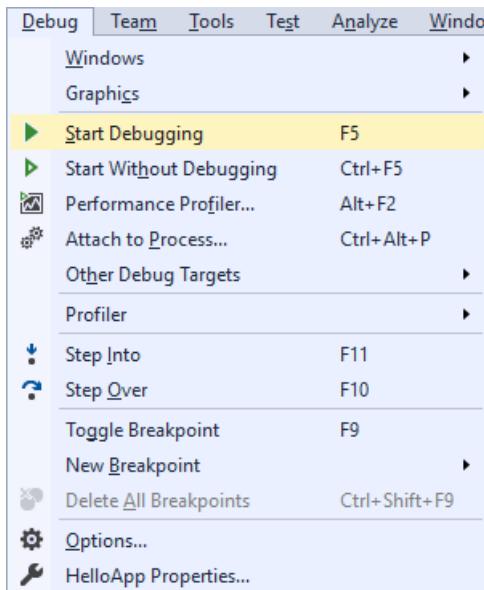


Debug and test the app

You can debug HelloApp to see whether the word "Hello" appears in the console window.

To debug the app

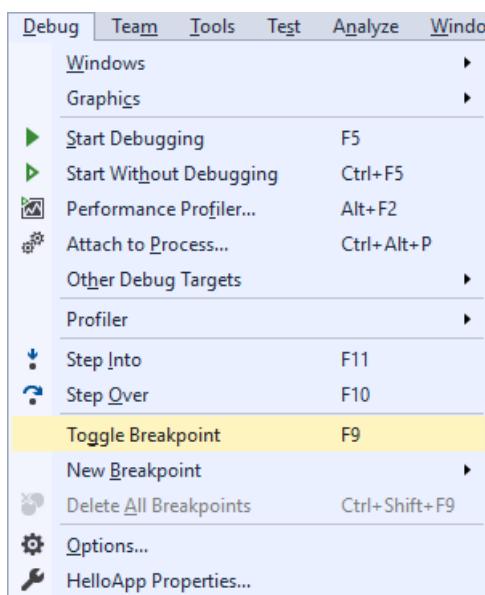
1. To start the debugger, choose **Debug > Start Debugging** on the menu bar.



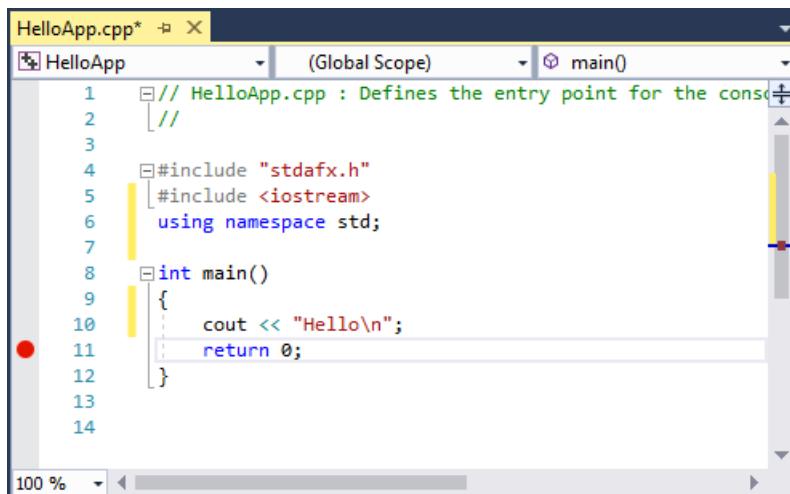
The debugger starts and runs the code. The console window (a separate window that looks like a command prompt) appears for a few seconds but closes quickly when the debugger stops running. To see the text, you need to set a breakpoint to stop program execution.

To add a breakpoint

1. In the editor, put the cursor on the line `return 0;`. On the menu bar, choose **Debug > Toggle Breakpoint**. You can also click in the left margin to set a breakpoint.

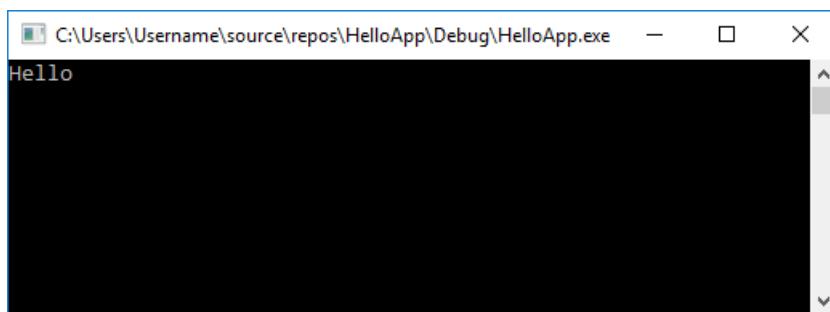


A red circle appears next to the line of code in the far left margin of the editor window.



2. To start debugging, press **F5**.

The debugger starts, and a console window appears showing the word **Hello**.



3. To stop debugging, press **Shift+F5**.

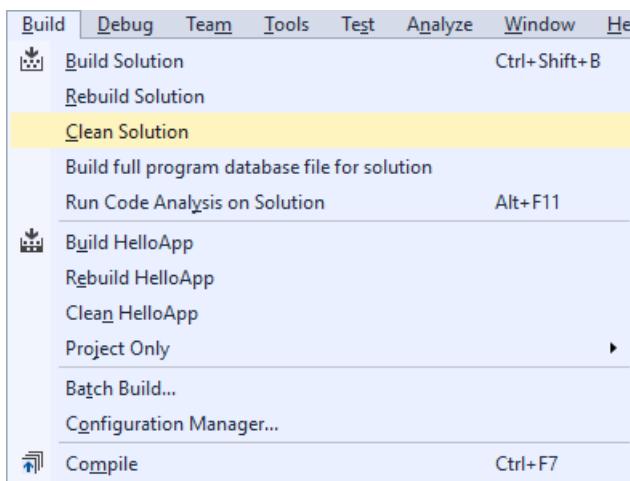
For more information about console project debugging, see [Console Projects](#).

Build a release version of the app

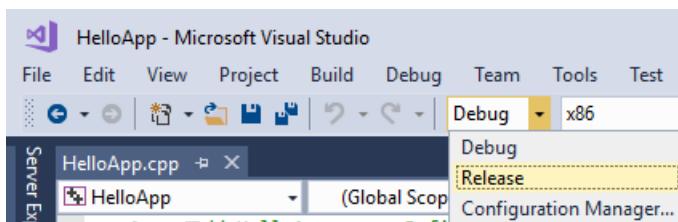
Now that you've verified that everything works, you can prepare a release build of the application. Release builds leave out the debugging information, and use compiler optimization options to create smaller, faster code.

To clean the solution files and build a release version

1. On the menu bar, choose **Build > Clean Solution** to delete intermediate files and output files that were created during previous builds.



2. To change the solution configuration for HelloApp from **Debug** to **Release**, in the toolbar, select the dropdown on the Solution Configurations control and then choose **Release**.



3. Build the solution. On the menu bar, choose **Build > Build Solution**.

When this build completes, you've created an app that you can copy and run in any command prompt window. It may not do much, but it's the gateway to greater things.

Congratulations on completing this quickstart! If you want to explore more examples, see [Visual Studio Samples](#).

See also

- [Using the Visual Studio IDE for C++ Desktop Development](#)
- [Walkthrough: Create a Simple Application with C# or Visual Basic](#)
- [Productivity Tips for Visual Studio](#)
- [Visual Studio Samples](#)
- [Get Started Developing with Visual Studio](#)

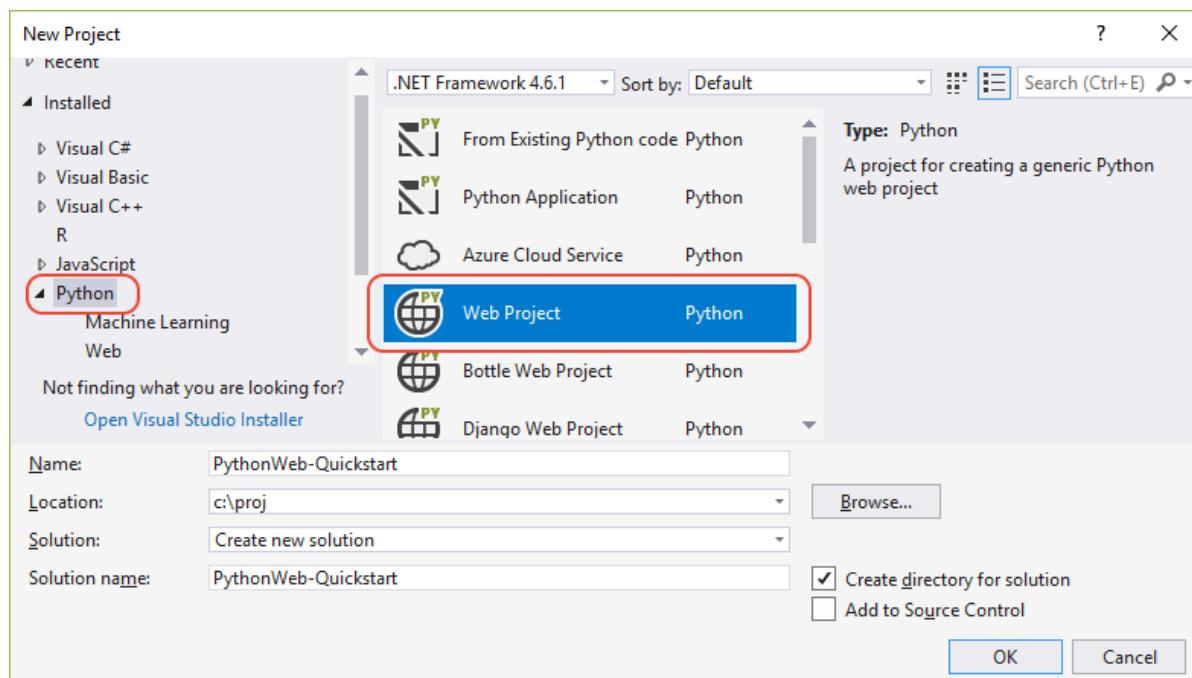
Quickstart: Use Visual Studio to create your first Python web app

3/16/2018 • 5 min to read • [Edit Online](#)

In this 5-10 minute introduction to the Visual Studio integrated development environment (IDE), you create a simple Python web application. If you haven't already installed Visual Studio, go to the [Visual Studio Downloads](#) page to install it for free. In the installer, make sure to select the **Python development** workload.

Create the project

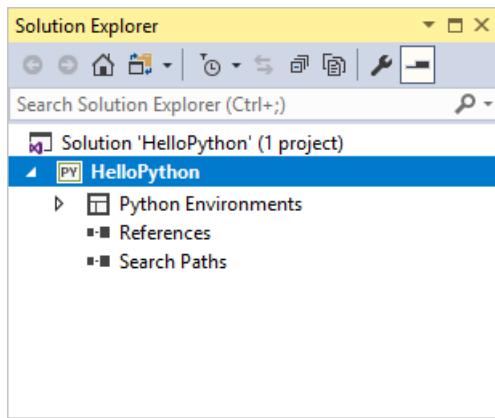
1. Open Visual Studio 2017.
2. From the top menu bar, choose **File > New > Project....**
3. In the **New Project** dialog box, in the left pane, expand **Installed**, then select **Python**.
4. In the middle pane, choose **Web project**, give the project a name like "HelloPython", then choose **OK**.



If you don't see the Python project templates, cancel out of the **New Project** dialog box and from the top menu bar, choose **Tools > Get Tools and Features...** to open the Visual Studio Installer. Choose the **Python development** workload, then choose **Modify**.



5. The new project opens in **Solution Explorer** in the right pane. The project is empty at this point because it contains no other files.



Question What's the advantage of creating a project in Visual Studio for a Python application?

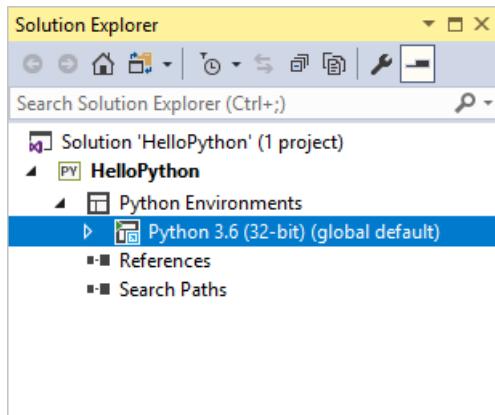
Answer: Python applications are typically defined using only folders and files, but this structure can become complex as applications become larger and perhaps involve auto-generated files, JavaScript for web applications, and so on. A Visual Studio project helps manage this complexity. The project (a `.pyproj` file) identifies all the source and content files associated with your project, contains build information for each file, maintains the information to integrate with source-control systems, and helps you organize your application into logical components.

Install the Falcon library

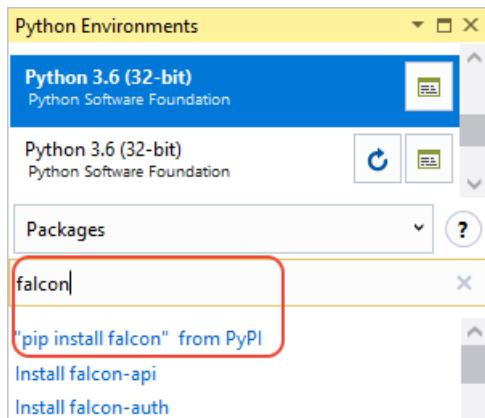
Web apps in Python almost always use one of the many available Python libraries to handle low-level details like routing web requests and shaping responses. For this purpose, Visual Studio provides a variety of templates for web apps using the Bottle, Flask, and Django frameworks.

In this Quickstart, however, you use the Falcon library to experience the process of installing a package and creating the web app from scratch. (Visual Studio does not at present include Falcon-specific templates.) For simplicity, the following steps install Falcon into the default global environment.

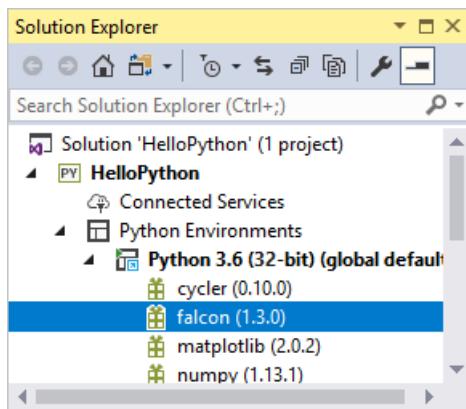
1. Expand the **Python Environments** node in the project to see the default environment for the project.



2. Right-click the environment and select **Install Python Package...**. This command opens the **Python Environments** window on the **Packages** tab. Enter "falcon" in the search field and select "**pip install falcon**" from PyPI. Accept any prompts for administrator privileges and observe the **Output** window in Visual Studio for progress. (A prompt for elevation happens when the packages folder for the global environment is located within a protected area like `c:\program files`.)



- Once installed, the library appears in the environment in **Solution Explorer**, which means that you can make use of it in Python code.



For more information about Falcon, visit falconframework.org.

Note that instead of installing libraries in the global environment, developers typically create a "virtual environment" in which to install libraries for a specific project. Many Python project templates in Visual Studio include a `requirements.txt` file that lists the libraries on which the template depends. Creating a project from one of those templates triggers creation of a virtual environment into which the libraries are installed. For more information, see [Using virtual environments](#).

Add a code file

You're now ready to add a bit of Python code to implement a minimal web app.

- Right-click the project in **Solution Explorer** and select **Add > New Item...**
- In the dialog that appears, select **Empty Python File**, name it `hello.py`, and select **Add**. Visual Studio automatically opens the file in an editor window. (In general, the **Add > New Item...** command is a great way to add different kinds of files to a project, as the item templates often provide useful boilerplate code.)
- Copy the following code and paste it into `hello.py`:

```

import falcon

# In Falcon, define a class for each resource and define on_* methods
# where * is any standard HTTP methods in lowercase, such as on_get.

class HelloResource:
    # In this application, the single HelloResource responds to only GET
    # requests, so it has only an on_get method.

    def on_get(self, req, resp):
        resp.status = falcon.HTTP_200 # 200 is the default
        resp.body = "Hello, Python!"

# Resources are represented by long-lived class instances
hello = HelloResource()

# Instruct Falcon to route / and /hello to HelloResource. If you add
# other resources, use api.add_route to define the desired
# resource locators for them.
api = falcon.API()
api.add_route('/', hello)
api.add_route('/hello', hello)

if __name__ == "__main__":
    # Use Python's built-in WSGI reference implementation to run
    # a web server for the application.
    from wsgiref.simple_server import make_server

    # Run the web server on localhost:8080
    print("Starting web app server")
    srv = make_server('localhost', 8080, api)
    srv.serve_forever()

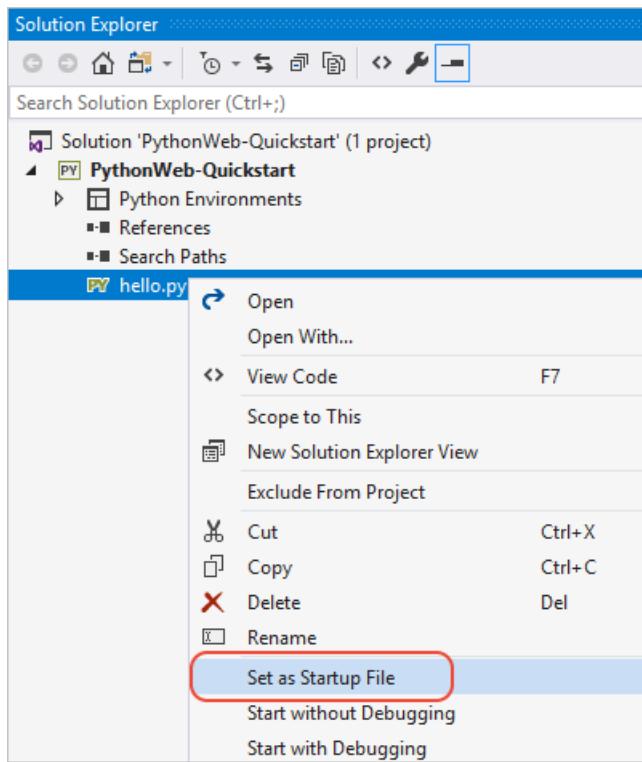
```

- After pasting this code, Visual Studio may show a squiggle underneath `falcon` in the first line and also underneath `wsgiref.simple.server` on line 20. These indicators appear when Visual Studio is still building the IntelliSense database for those modules.

For more information about Falcon, see the [Falcon Quickstart](#) (`falcon.readthedocs.io`).

Run the application

- Right-click `hello.py` in **Solution Explorer** and select **Set as startup file**. The command identifies the code file to launch in Python when running the app.



2. Right-click the "Hello Python" project in **Solution Explorer** and select **Properties**. Then select the **Debug** tab and set the **Port Number** property to `8080`. This step ensures that Visual Studio launches a browser with `localhost:8080` rather than using a random port.

3. Select **Debug > Start Without Debugging** (Ctrl+F5) to save changes to files and run the app.

4. A command window appears with the message "Starting web app server", then a browser window should open to `localhost:8080` where you see the message, "Hello, Python!" The GET request also appear in the command window.

If a browser does not open automatically, start the browser of your choice and navigate to `localhost:8080`.)

If you see only the Python interactive shell in the command window, or if that window flashes on the screen briefly, ensure that you set `hello.py` as the startup file in step 1 above.

5. Close the command window to stop the app, then close the browser window.

Next steps

Congratulations on completing this Quickstart, in which you've learned a little about the Visual Studio IDE with Python. To continue with a fuller tutorial on Python in Visual Studio, including using the interactive window, debugging, data visualization, and working with Git, select the button below.

[Tutorial: Getting Started with Python in Visual Studio.](#)

- Learn about [Python web app templates in Visual Studio](#)
- Learn about [Python debugging](#)
- Learn more about the [Visual Studio IDE](#)

Quickstart: Use Visual Studio to create your first Node.js app

3/15/2018 • 2 min to read • [Edit Online](#)

In this 5-10 minute introduction to the Visual Studio integrated development environment (IDE), you'll create a simple Node.js web application. If you haven't already installed Visual Studio, install it for free [here](#).

Create a project

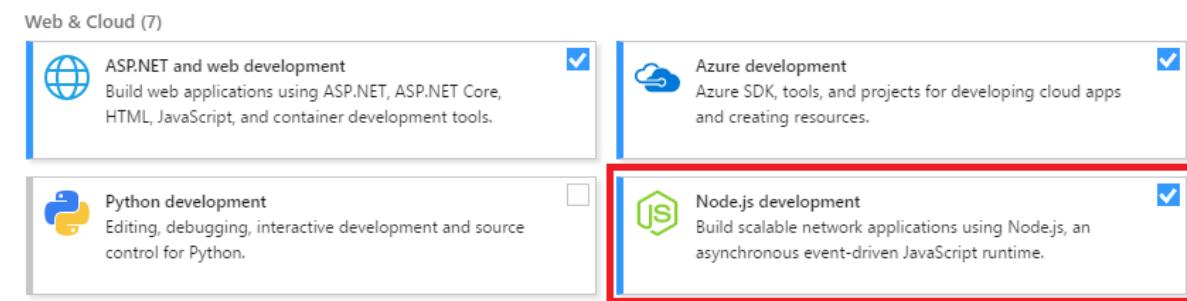
First, you'll create an Node.js web application project.

1. If you don't have the Node.js runtime already installed, install the LTS version from the [Node.js](#) website.

In general, Visual Studio automatically detects the installed Node.js runtime. If it does not detect an installed runtime you can configure your project to reference the installed runtime.

2. Open Visual Studio 2017.
3. From the top menu bar, choose **File > New > Project....**
4. In the **New Project** dialog box, in the left pane, expand **JavaScript**, then choose **Node.js**. In the middle pane, choose **Blank Node.js Web application**, then choose **OK**.

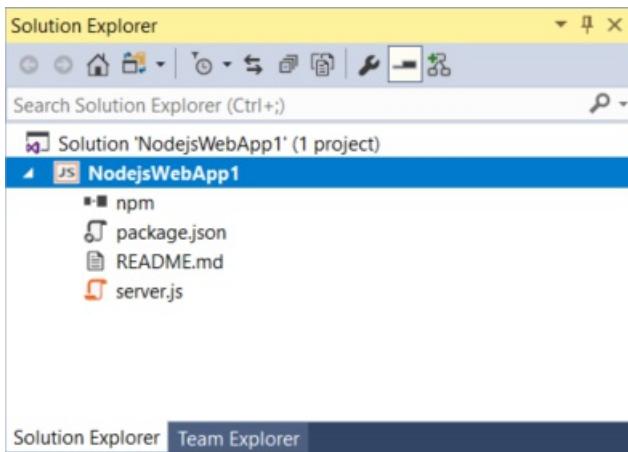
If you don't see the **Blank Node.js Web application** project template, click the **Open Visual Studio Installer** link in the left pane of the **New Project** dialog box. The Visual Studio Installer launches. Choose the **Node.js development** workload, then choose **Modify**.



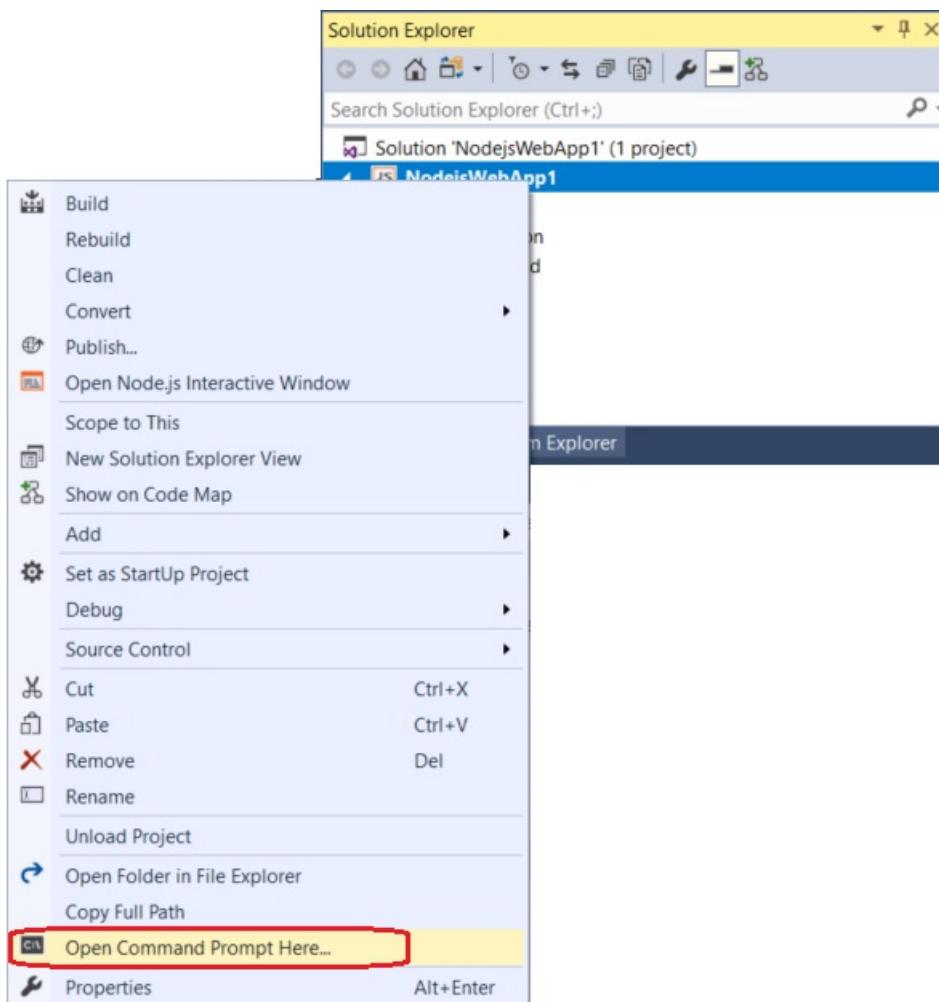
Visual Studio creates and the new solution and opens the project. `server.js` opens in the editor in the left pane.

Explore the IDE

1. Take a look at **Solution Explorer** in the right pane.



- Highlighted in bold is your project, using the name you gave in the **New Project** dialog box. On disk, this project is represented by a .njsproj file in your project folder.
 - At the top level is a solution, which by default has the same name as your project. A solution, represented by a .sln file on disk, is a container for one or more related projects.
 - The npm node shows any installed npm packages. You can right-click the npm node to search for and install npm packages using a dialog box.
2. If you want to install npm packages or node.js commands from a command prompt, right-click the project node and choose **Open Command Prompt Here**.



3. In the `server.js` file in the editor (left pane), choose `http.createServer` and then press **F12** or choose **Go To Definition** from the context (right-click) menu. This command takes you to the definition of the `createServer` function in `index.d.ts`.

```

server.js  X
JS NodejsWebApp1  <global>
1  'use strict';
2  var http = require('http');
3  var port = process.env.PORT || 1337;
4
5  http.createServer(function (req, res) {
6      res.writeHead(200, { 'Content-Type': 'text/plain' });
7      res.end('Hello World\n' + res.connection.
8  }).listen(port)
9

```

The screenshot shows the Visual Studio Code interface with the file 'server.js' open. A context menu is displayed over the line of code 'res.end('Hello World\n' + res.connection.'. The menu item 'Go To Definition' is highlighted in yellow.

4. Got back to *server.js*, then put your cursor at the end of the string in this line of code, `res.end('Hello World\n');`, and modify it so that it looks like this:

```
res.end('Hello World\n' + res.connection.
```

Where you type `connection.`, IntelliSense provides options to auto-complete the code entry.

```

server.js*  X
JS NodejsWebApp1  <function>
1  'use strict';
2  var http = require('http');
3  var port = process.env.PORT || 1337;
4
5  http.createServer(function (req, res) {
6      res.writeHead(200, { 'Content-Type': 'text/plain' });
7      res.end('Hello World\n' + res.connection.
8  }).listen(port)
9

```

The screenshot shows the Visual Studio Code interface with the file 'server.js*' open. The cursor is at the end of the string 'res.end('Hello World\n' + res.connection.'. An auto-completion dropdown is open, showing various properties of the 'connection' object, with 'localPort' highlighted.

5. Choose **localPort**, and then type `);` to complete the statement so that it looks like this:

```
res.end('Hello World\n' + res.connection.localPort);
```

Run the application

1. Press **Ctrl+F5** (or **Debug > Start Without Debugging**) to run the application. The app opens in a browser.
2. In the browser window, you will see "Hello World" plus the local port number.
3. Close the web browser.

Congratulations on completing this quickstart! We hope you learned a little bit about the Visual Studio IDE. If you'd like to delve deeper into its capabilities, please continue with a tutorial in the **Tutorials** section of the table of contents.

Next steps

- Go through the [Tutorial for Node.js and Express](#)
- Go through the [Tutorial for Node.js and React](#)

Quickstart: Use Visual Studio to create your first ASP.NET Core web app

3/15/2018 • 3 min to read • [Edit Online](#)

In this 5-10 minute introduction to the Visual Studio integrated development environment (IDE), you'll create a simple C# ASP.NET Core web application.

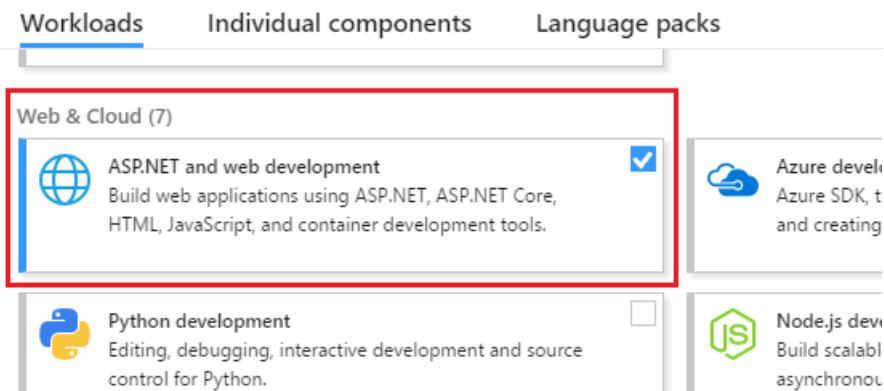
If you haven't already installed Visual Studio, go to the [Visual Studio Downloads](#) page to install it for free.

Create a project

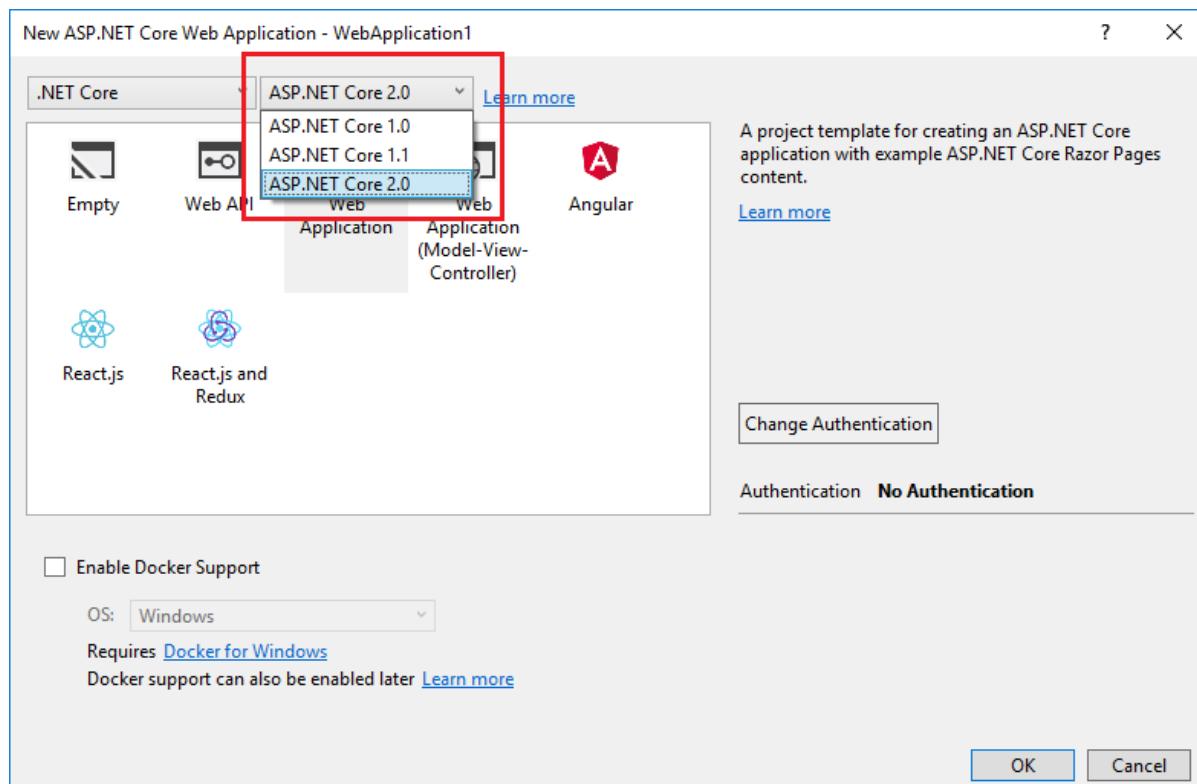
First, you'll create an ASP.NET Core web application project. The project type comes with template files that constitute a functional web application, before you've even added anything!

1. Open Visual Studio 2017.
2. From the top menu bar, choose **File > New > Project...**
3. In the **New Project** dialog box, in the left pane, expand **Visual C#**, then choose **.NET Core**. In the middle pane, choose **ASP.NET Core Web Application**, then choose **OK**.

If you don't see the **.NET Core** project template category, choose the [Open Visual Studio Installer](#) link in the left pane. The Visual Studio Installer launches. Choose the **ASP.NET and web development** workload, then choose **Modify**.

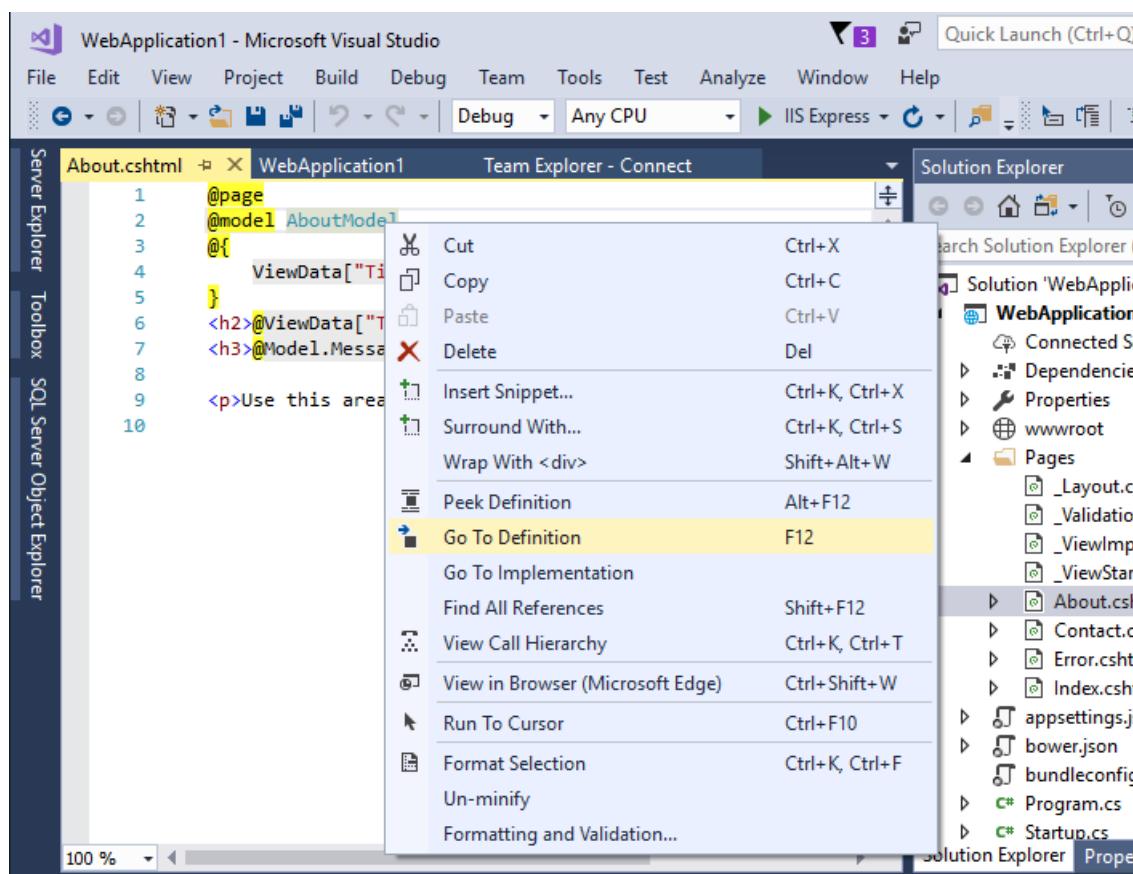


4. In the **New ASP.NET Core Web Application** dialog box, select **ASP.NET Core 2.0** from the top drop-down menu. (If you don't see **ASP.NET Core 2.0** in the list, install it by following the [Download](#) link that should appear in a yellow bar near the top of the dialog box.) Choose **OK**.



Explore the IDE

1. In the **Solution Explorer** toolbar, expand the **Pages** folder, then choose **About.cshtml** to open it in the editor. This file corresponds to a page called **About** in the web application.
2. In the editor, choose **AboutModel** and then press **F12** or choose **Go To Definition** from the context (right-click) menu. This command takes you to the definition of the **AboutModel** C# class.



3. Next we'll clean up the **using** directives at the top of the file using a simple shortcut. Choose any of the

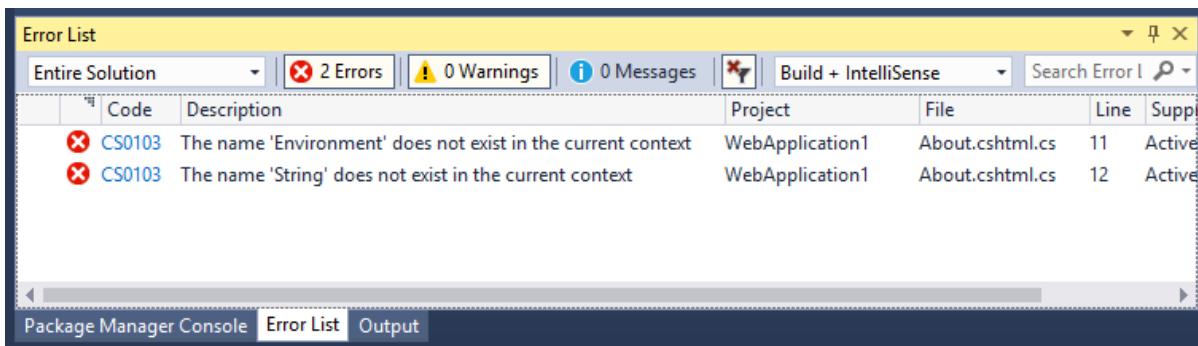
greyed-out using directives and a **Quick Actions** light bulb will appear just below the caret or in the left margin. Choose the light bulb, and then choose **Remove Unnecessary Usings**.

The unnecessary usings are deleted from the file.

4. In the `OnGet()` method, change the body to the following code:

```
public void OnGet()
{
    string directory = Environment.CurrentDirectory;
    Message = String.Format("Your directory is {0}.", directory);
}
```

5. You'll see two wavy underlines appear under **Environment** and **String**, because these types are not in scope. Open the **Error List** toolbar to see the same errors listed there. (If you don't see the **Error List** toolbar, choose **View > Error List** from the top menu bar.)



6. In the editor window, place your cursor on either line that contains the error, then choose the Quick Actions light bulb in the left margin. From the drop-down menu, choose **using System**; to add this directive to the top of your file and resolve the errors.

Run the application

1. Press **Ctrl+F5** to run the application and open it in a web browser.
2. At the top of the web site, choose **About** to see the directory message you added in the `OnGet()` method for the **About** page.
3. Close the web browser.

NOTE

If you get an error message that says **Unable to connect to web server 'IIS Express'**, close Visual Studio and then open it using the **Run as administrator** option from the right-click or context menu. Then, run the application again.

Congratulations on completing this quickstart! We hope you learned a little bit about the Visual Studio IDE. If you'd like to delve deeper into its capabilities, please continue with a tutorial in the **Tutorials** section of the table of contents.

Next Steps

Congratulations on completing this Quickstart! We hope you learned a little bit about C#, ASP.NET Core, and the Visual Studio IDE. To learn more, continue with the following tutorial.

[Getting started with C# and ASP.NET in Visual Studio](#)

Quickstart: Create your first console app in Visual Studio with Visual Basic

3/15/2018 • 1 min to read • [Edit Online](#)

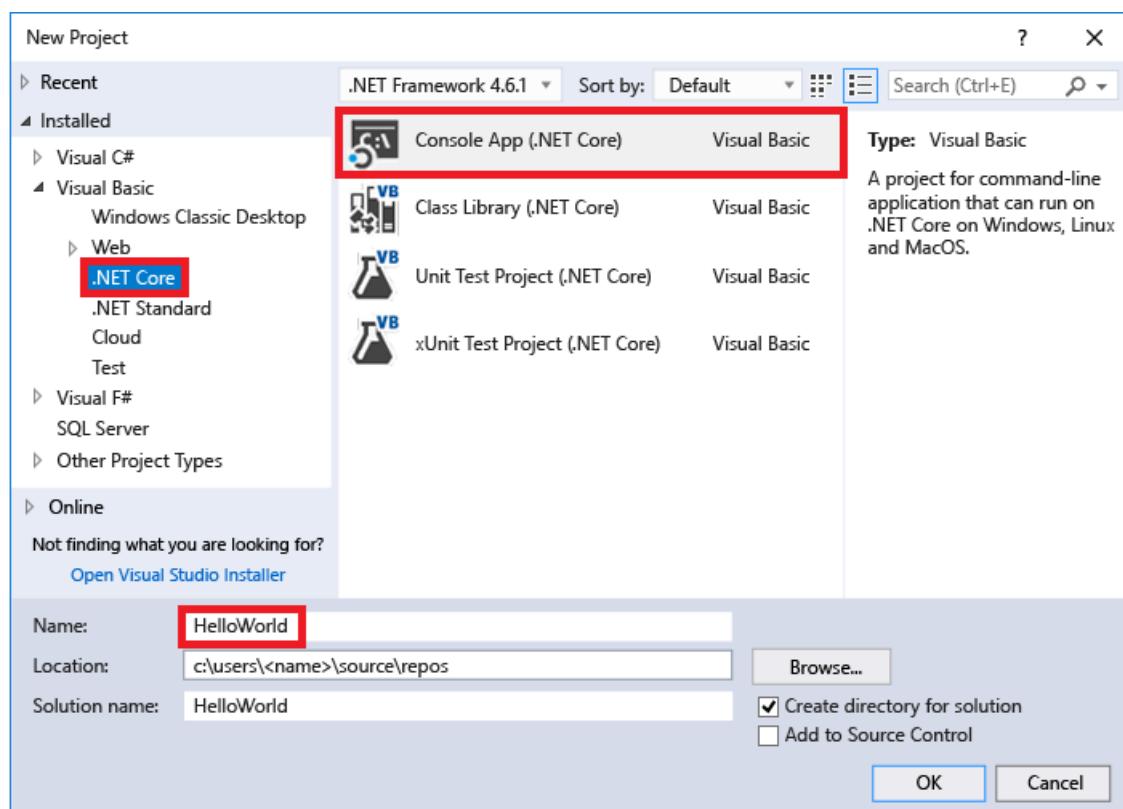
In this 5-10 minute introduction to the Visual Studio integrated development environment (IDE), you'll create a simple Visual Basic application that runs on the console.

If you haven't already installed Visual Studio, go to the [Visual Studio downloads](#) page to install it for free.

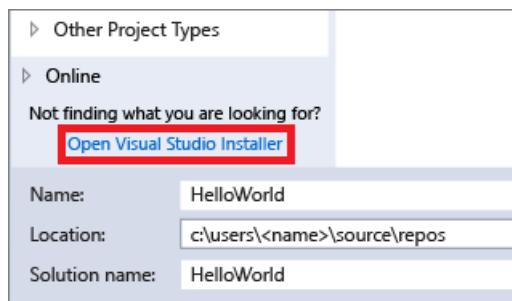
Create a project

First, you'll create a Visual Basic application project. The project type comes with all the template files you'll need, before you've even added anything!

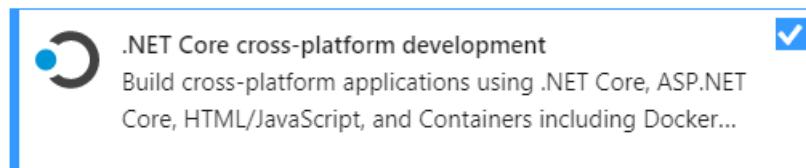
1. Open Visual Studio 2017.
2. From the top menu bar, choose **File > New > Project...**
3. In the **New Project** dialog box in the left pane, expand **Visual Basic**, and then choose **.NET Core**. In the middle pane, choose **Console App (.NET Core)**. Then name the project *HelloWorld*.



If you don't see the **Console App (.NET Core)** project template, click the [Open Visual Studio Installer](#) link in the left pane of the **New Project** dialog box.



The Visual Studio Installer launches. Choose the **.NET Core cross-platform development** workload, and then choose **Modify**.

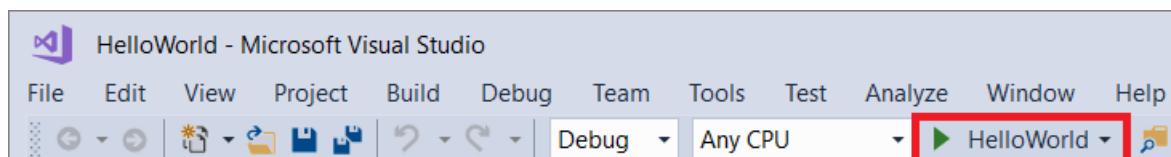


Create the application

After you select your Visual Basic project template and name your project, Visual Studio creates a simple "Hello World" application for you. It calls the [WriteLine](#) method to display the literal string "Hello World!" in the console window.

```
Program.vb  X
VB HelloWorld  Program
1 Imports System
2
3 Module Program
4 Sub Main(args As String())
5     Console.WriteLine("Hello World!")
6 End Sub
7 End Module
8
```

If you click the **HelloWorld** button in the IDE, you can run the program in Debug mode.



When you do this, the console window is visible for only a moment before it closes. This happens because the [Main](#) method terminates after its single statement executes, and so the application ends.

Add some code

Let's add some code to pause the application and then ask for user input.

1. Add the following code immediately after the call to the [WriteLine](#) method:

```
Console.Write("Press any key to continue...")
Console.ReadKey(true)
```

This pauses the program until you press a key.

2. On the menu bar, select **Build > Build Solution**.

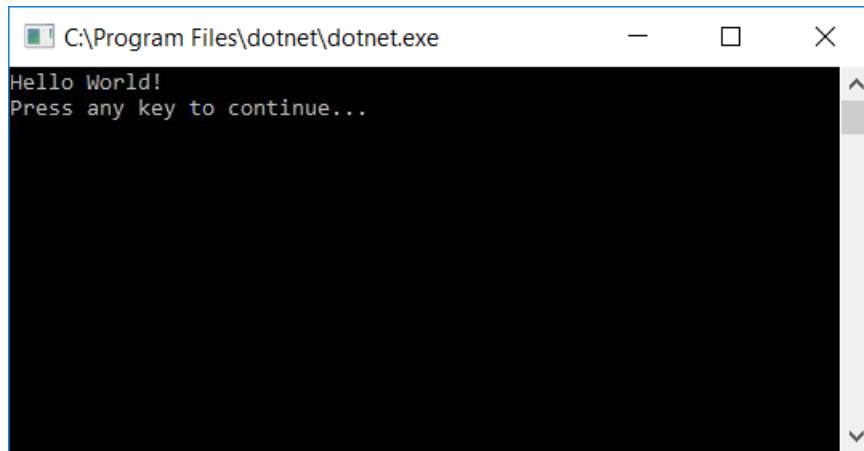
This compiles your program into an intermediate language (IL) that's converted into binary code by a just-in-time (JIT) compiler.

Run the application

1. Click the **HelloWorld** button on the toolbar.



2. Press any key to close the console window.



Next Steps

Congratulations on completing this Quickstart! We hope you learned a little bit about Visual Basic and the Visual Studio IDE. To learn more, continue with the following tutorial.

[Getting started with Visual Basic in Visual Studio](#)

Working with Python in Visual Studio

3/27/2018 • 3 min to read • [Edit Online](#)

Python is a popular programming language that is reliable, flexible, easy to learn, free to use on all operating systems, and supported by both a strong developer community and many free libraries. The language supports all manners of development, including web applications, web services, desktop apps, scripting, and scientific computing and is used by many universities, scientists, casual developers, and professional developers alike.

Visual Studio provides first-class language support for Python. This tutorial guides you through the following steps:

- [Step 0: Installation](#)
- [Step 1: Creating a Python project \(this article\)](#)
- [Step 2: Writing and running code to see Visual Studio IntelliSense at work](#)
- [Step 3: Create more code in the interactive REPL window](#)
- [Step 4: Run the completed program in the Visual Studio debugger](#)
- [Step 5: Installing packages and managing Python environments](#)
- [Step 6: Working with Git](#)

Prerequisites

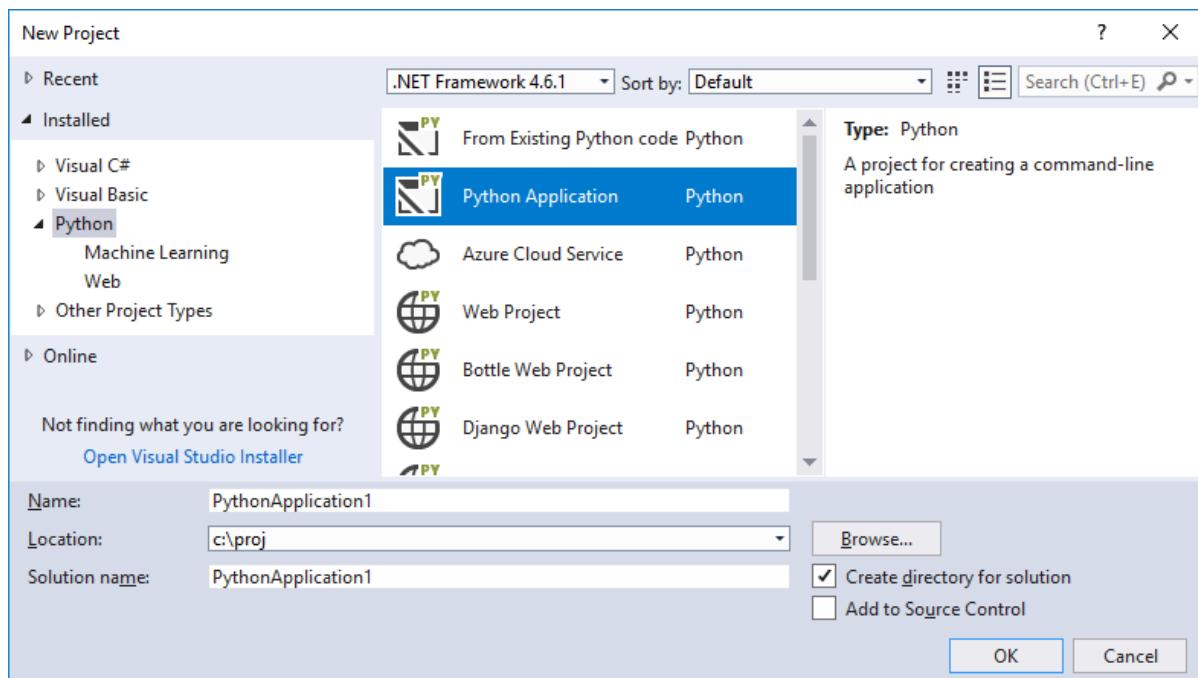
- Visual Studio 2017 with the Python workload installed. See [Step 0](#) for instructions.

Step 1: Create a new Python project

A *project* is how Visual Studio manages all the files that come together to produce a single application, including source code, resources, configurations, and so on. A project formalizes and maintains the relationship between all the project's files as well as external resources that are shared between multiple projects. As such, project allow your application to effortlessly expand and grow much easier than simply managing a project's relationships in ad hoc folders, scripts, text files, and even your own mind.

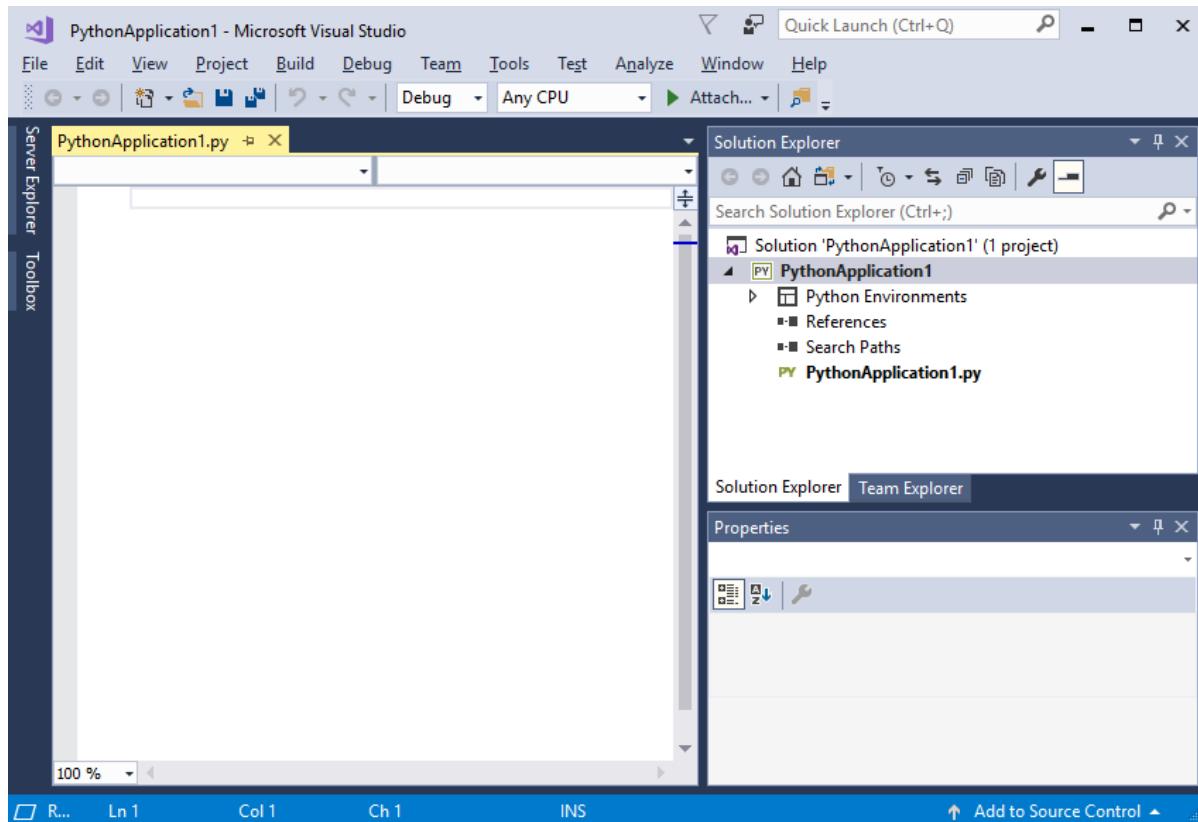
In this tutorial you begin with a simple project containing a single, empty code file.

1. In Visual Studio, select **File > New > Project** (Ctrl+Shift+N), which brings up the **New Project** dialog. Here you browse templates across different languages, then select one for your project and specify where Visual Studio places files.
2. To view Python templates, select **Installed > Python** on the left, or searching for "Python". Using search is a great way to find a template when you can't remember its location in the languages tree.

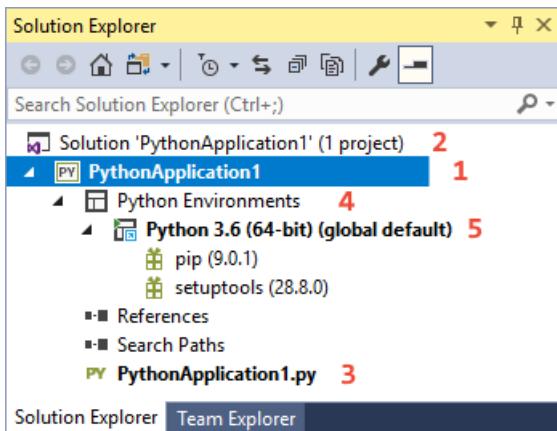


Notice how Python support in Visual Studio includes a number of project templates, including web applications using the Bottle, Flask, and Django frameworks. For the purposes of this walkthrough, however, let's start with an empty project.

3. Select the **Python Application** template, specify a name for the project, and select **OK**.
4. After a few moments, Visual Studio shows the project structure in the **Solution Explorer** window (1). The default code file is open in the editor (2). The properties window (3) also appears to show additional information for any item selected in Solution Explorer, including its exact location on disk.



5. Take a few moments to familiarize yourself with Solution Explorer, which is where you browse files and folders in your project.



- (1) Highlighted in bold is your project, using the name you gave in the New Project dialog. On disk, this project is represented by a `.pyproj` file in your project folder.
- (2) At the top level is a *solution*, which by default has the same name as your project. A solution, represented by a `.sln` file on disk, is a container for one or more related projects. For example, if you write a C++ extension for your Python application, that C++ project could reside within the same solution. The solution might also contain a project for a web service, along with projects for dedicated test programs.
- (3) Under your project you see source files, in this case only a single `.py` file. Selecting a file displays its properties in the Properties window. Double-clicking a file opens it in whatever way is appropriate for that file.
- (4) Also under the project is the **Python Environments** node. When expanded, you see the Python interpreters that are available to you. Expand an interpreter node to see the libraries that are installed into that environment (5).

Right-click any node or item in Solution Explorer to access a menu of applicable commands. For example, the **Rename** command allows you to change the name of any node or item, including the project and the solution.

Next steps

[Writing and running code](#)

Going deeper

- [Python projects in Visual Studio](#).
- [Learn about the Python language on python.org](#)
- [Python for Beginners \(python.org\)](#)
- [Free Python courses on Microsoft Virtual Academy](#)
- [Top Python Questions at Microsoft Virtual Academy](#)

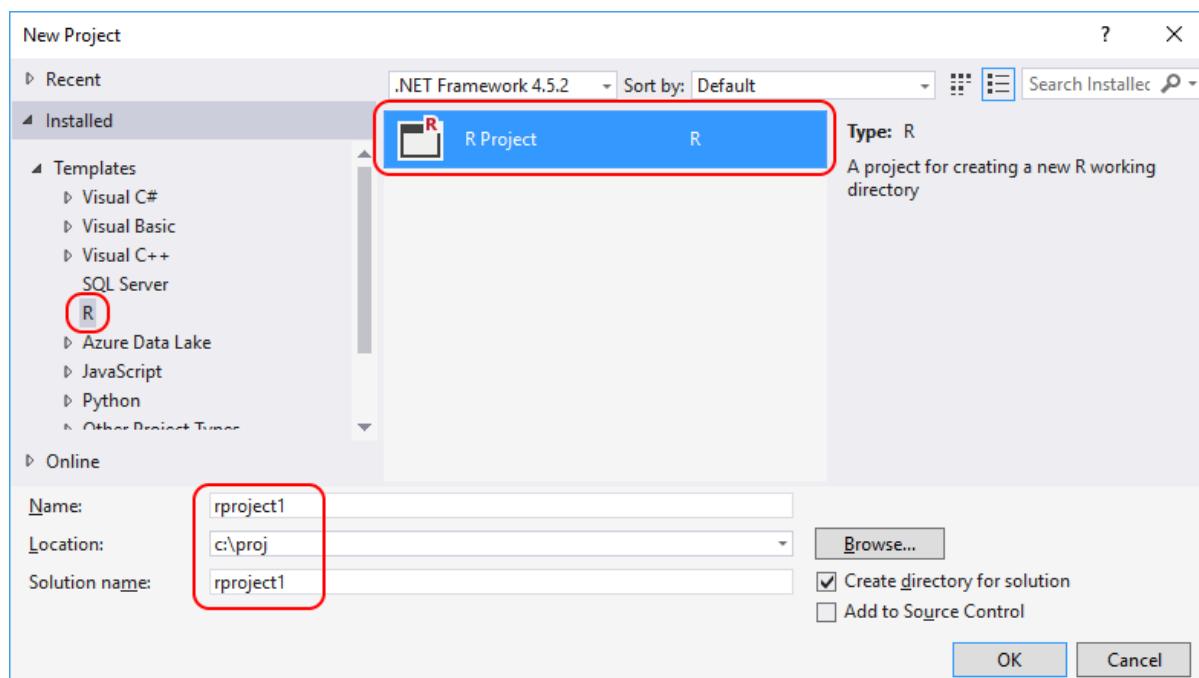
Getting started with R Tools for Visual Studio

2/8/2018 • 7 min to read • [Edit Online](#)

Once you have R Tools for Visual Studio (RTVS) installed (see [Installation](#)), you can quickly get a taste of the experience that those tools provide.

Create an R project

1. Start Visual Studio.
2. Choose **File > New > Project...** (Ctrl+Shift+N)
3. Select "R Project" from under **Templates > R**, give the project a name and location, and select **OK**:



4. Once the project is created, you see the following windows:

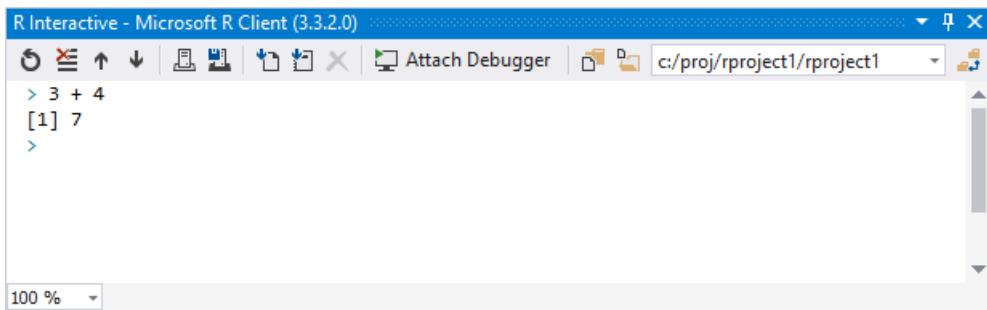
- On the right is Visual Studio Solution Explorer, where you see your project inside a containing *solution*. (Solutions can contain any number of projects of different types; see [Projects](#) for details.)
- On the top left is a new R file (`script.R`) where you can edit source code with all of Visual Studio's editing features.
- On the bottom left is the **R Interactive** window in which you can interactively develop and test code.

NOTE

You can use the **R Interactive** window without having any projects open, and even when a different project type is loaded. Just select **R Tools > Windows > R Interactive** at any time.

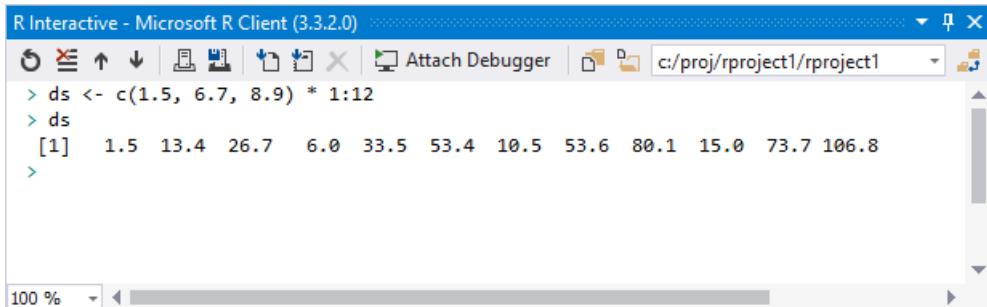
Explore the Interactive Window and IntelliSense

1. Test that the interactive window is working by typing in `3 + 4` and then Enter to see the result:



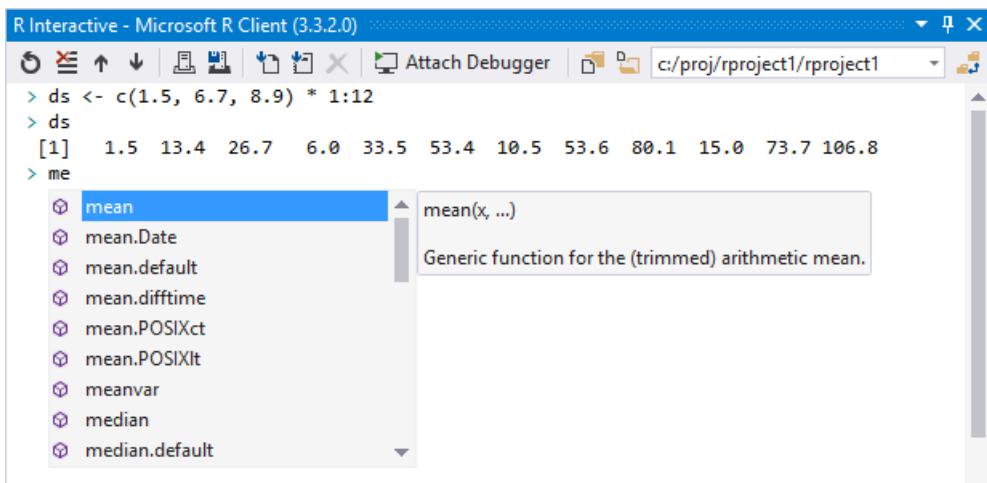
```
R Interactive - Microsoft R Client (3.3.2.0)
> 3 + 4
[1] 7
```

2. Enter something a little more complicated, `ds <- c(1.5, 6.7, 8.9) * 1:12`, and then enter `ds` to see the result:



```
R Interactive - Microsoft R Client (3.3.2.0)
> ds <- c(1.5, 6.7, 8.9) * 1:12
> ds
[1] 1.5 13.4 26.7 6.0 33.5 53.4 10.5 53.6 80.1 15.0 73.7 106.8
```

3. Type in `mean(ds)` but notice that as soon as you type `m` or `me`, Visual Studio IntelliSense provides auto-completion options. When the completion you want is selected in the list, press Tab to insert it; you can change the selection with the arrow keys or the mouse.

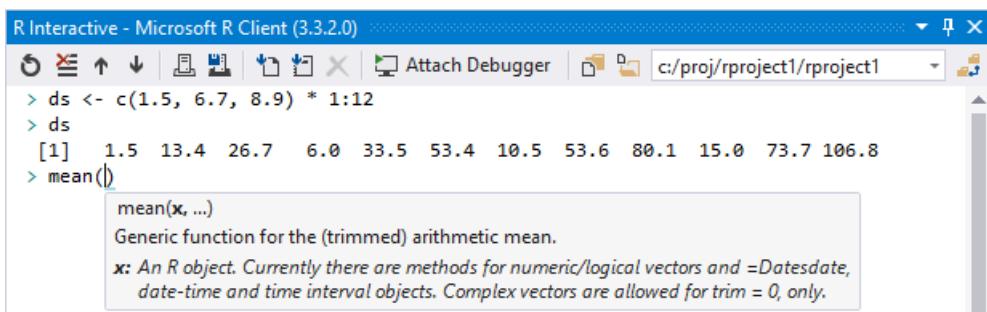


```
R Interactive - Microsoft R Client (3.3.2.0)
> ds <- c(1.5, 6.7, 8.9) * 1:12
> ds
[1] 1.5 13.4 26.7 6.0 33.5 53.4 10.5 53.6 80.1 15.0 73.7 106.8
> me
    mean
    mean.Date
    mean.default
    mean.difftime
    mean.POSIXct
    mean.POSIXlt
    meanvar
    median
    median.default
```

mean
mean.Date
mean.default
mean.difftime
mean.POSIXct
mean.POSIXlt
meanvar
median
median.default

mean(x, ...)
Generic function for the (trimmed) arithmetic mean.

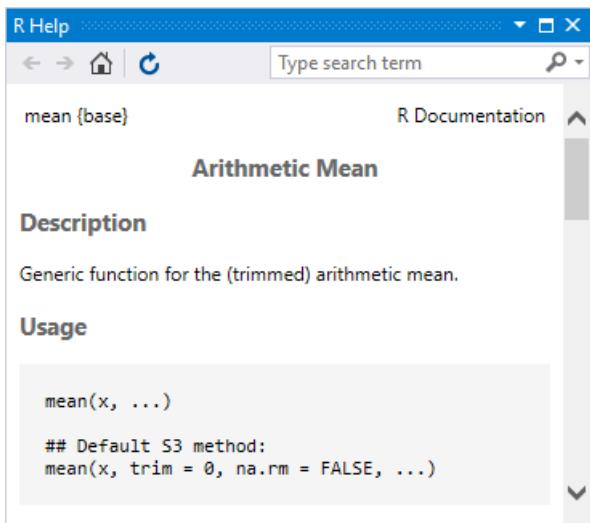
4. After completing `mean`, type the opening parenthesis `(` and note how IntelliSense gives you inline help for the function:



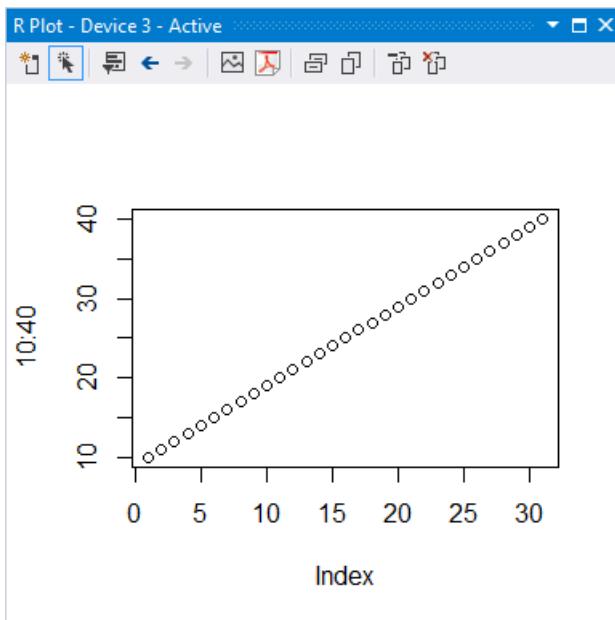
```
R Interactive - Microsoft R Client (3.3.2.0)
> ds <- c(1.5, 6.7, 8.9) * 1:12
> ds
[1] 1.5 13.4 26.7 6.0 33.5 53.4 10.5 53.6 80.1 15.0 73.7 106.8
> mean()
    mean(x, ...)
    Generic function for the (trimmed) arithmetic mean.
    x: An R object. Currently there are methods for numeric/logical vectors and =Datesdate,
        date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
```

5. Complete the line `mean(ds)` and press Enter to see the result (`[1] 39.51667`).

6. The Interactive window is integrated with help, so entering `?mean` displays help for that function in the **R Help** window in Visual Studio. For details, see [Help in R Tools for Visual Studio](#).



7. Some commands, such as `plot(1:100)`, open a new window in Visual Studio when the output can't be displayed directly in the interactive window:



The interactive window also lets you review your history, load and save workspaces, attach to a debugger, and interact with source code files instead of using copy-paste. See [Working with the R Interactive Window](#) for details.

Experience code editing features

Working briefly with the interactive window demonstrates basic editing features like IntelliSense that also work in the code editor. If you enter the same code as before, you see the same auto-completion and IntelliSense prompts, but not the output.

Writing code in a `.R` file lets you see all your code at once, and makes it easier to make small changes and then quickly see the result by running the code in the interactive window. You can also have as many files as you want in a project. When code is in a file, you can also run it step-by-step in the debugger (discussed later in this article). These capabilities are helpful when you're developing computational algorithms and writing code to manipulate one or more datasets, especially when you want to examine all intermediate results.

As an example, the following steps create a little code to explore the [Central Limit Theorem](#) (Wikipedia). (This example is adapted from the *R Cookbook* by Paul Teator.)

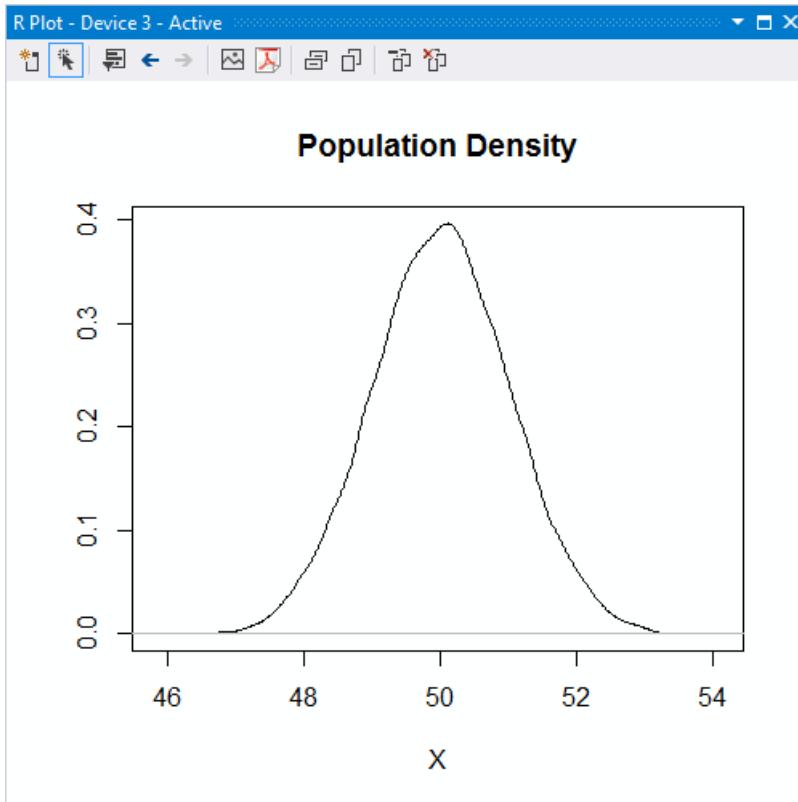
1. In the `script.R` editor, enter the following code:

```

mu <- 50
stddev <- 1
N <- 10000
pop <- rnorm(N, mean = mu, sd = stddev)
plot(density(pop), main = "Population Density", xlab = "X", ylab = "")

```

2. To quickly see the results, select all the code (Ctrl+A), then press Ctrl+Enter or right-click and select **Execute In Interactive**. All the selected code is run in the interactive window as if you typed it directly, showing the result in a plot window:



3. For a single line, just press Ctrl+Enter at any time to run that line in the interactive window.

TIP

Learn the pattern of making edits and pressing Ctrl+Enter (or selecting everything with Ctrl+A and then pressing Ctrl+Enter) to quickly run the code. Doing so is much more efficient than using the mouse for the same operations.

In addition, you can drag and drop the plot window out of the Visual Studio frame and place it whenever else you want on your display. You can then resize the plot window to the dimensions you want and then save it to an image or PDF file.

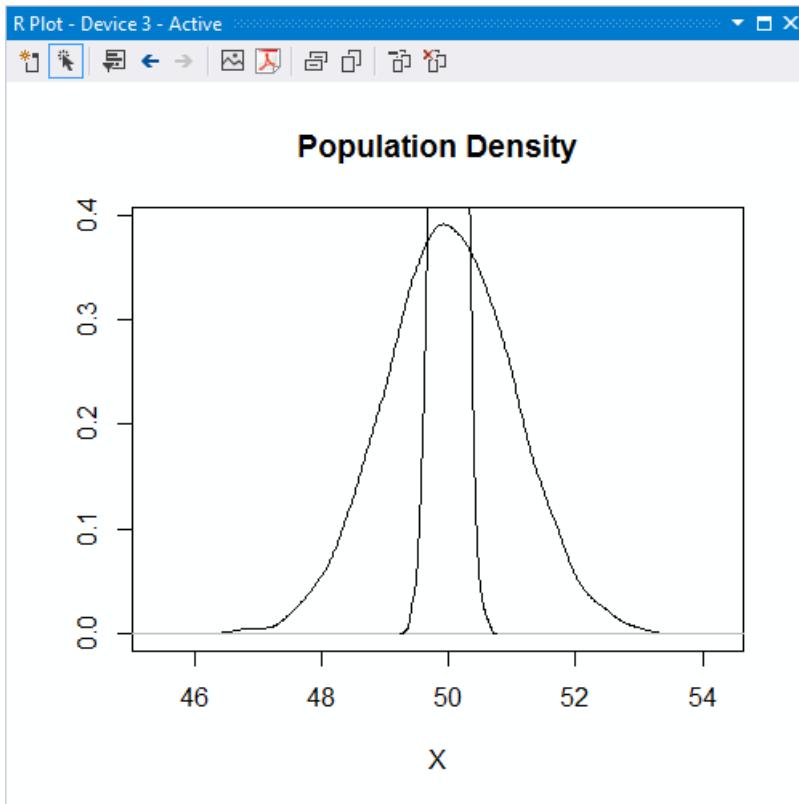
1. Add a few more lines of code to include a second plot:

```

n <- 30
samp.means <- rnorm(N, mean = mu, sd = stddev / sqrt(n))
lines(density(samp.means))

```

2. Press Ctrl+A and Ctrl+Enter again to run the code, producing the following result:



3. The problem is that the first plot determines the vertical scale, so the second plot (with `lines`) doesn't fit. To correct this problem, we need to set the `ylim` parameter on the `plot` call, but do so properly we need to add code to calculate the maximum vertical value. Doing this line-by-line in the interactive window is inconvenient because we need to rearrange the code to use `samp.means` before calling `plot`. In a code file, though, we can easily make the appropriate edits:

```

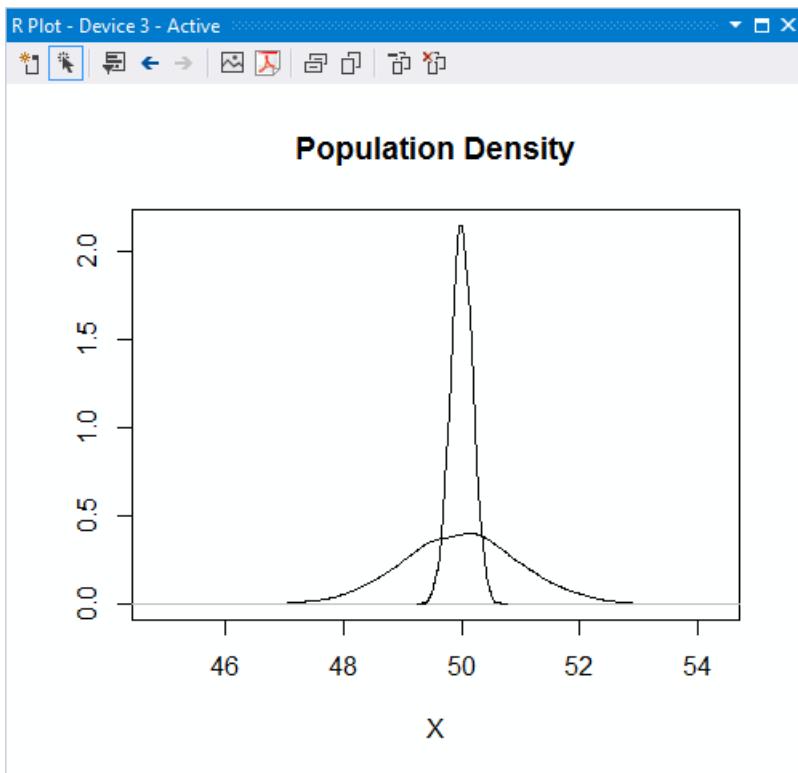
mu <- 50
stddev <- 1
N <- 10000
pop <- rnorm(N, mean = mu, sd = stddev)

n <- 30
samp.means <- rnorm(N, mean = mu, sd = stddev / sqrt(n))
max.samp.means <- max(density(samp.means)$y)

plot(density(pop), main = "Population Density",
      ylim = c(0, max.samp.means),
      xlab = "X", ylab = "")
lines(density(samp.means))

```

4. Ctrl+A and Ctrl+Enter again to see the result:



There's more you can do in the editor. For details, see [editing code](#), [IntelliSense](#), and [code snippets](#).

Debugging your code

One of the key strengths of Visual Studio is its debugging UI. RTVS builds on top of this strong foundation and adds innovative UI such as the [Variable Explorer](#) and [Data Table Viewer](#). Here, let's just take a first look at debugging.

1. To begin, reset the current workspace to clear everything you've done so far by using the **R Tools > Session > Reset** menu command. By default, everything you do in the interactive window accrues to the current session, which is then also used by the debugger. By resetting the session, you ensure that the debugging session starts with no pre-existing data. The **Reset** command, however, doesn't affect your `script.R` source file, because that's managed and saved outside of the workspace.
2. With the `script.R` file created in the previous section, set a breakpoint on the line that begins with `pop <-` by placing the caret on that line and then pressing F9, or selecting the **Debug > Toggle Breakpoint** menu command. Alternately, simply click in the left-hand margin (or gutter) for that line where the red breakpoint dot appears:

```

script.R + X
mu <- 50
stddev <- 1
N <- 10000
pop <- rnorm(N, mean = mu, sd = stddev)

n <- 30
samp.means <- rnorm(N, mean = mu, sd = stddev / sqrt(n))
max.samp.means <- max(density(samp.means)$y)

plot(density(pop), main = "Population Density",
      ylim = c(0, max.samp.means),
      xlab = "X", ylab = "")
lines(density(samp.means))

```

3. Launch the debugger with the code in `script.R` by either selecting the **Source startup file** button on the toolbar, selecting the **Debug > Source startup file** menu items, or pressing F5. Visual Studio enters its debugging mode and starts running the code. It stops, however, on the line where you set the breakpoint:

```

script.R  X
mu <- 50
stddev <- 1
N <- 10000
pop <- rnorm(N, mean = mu, sd = stddev)

n <- 30
samp.means <- rnorm(N, mean = mu, sd = stddev / sqrt(n))
max.samp.means <- max(density(samp.means)$y)

plot(density(pop), main = "Population Density",
      ylim = c(0, max.samp.means),
      xlab = "X", ylab = "")
lines(density(samp.means))

```

4. During debugging, Visual Studio provides the ability to step through your code line by line. You can also step into functions, step over them, or step out of them to the calling context. These capabilities, along with others, can be found on the **Debug** menu, the right-click context menu in the editor, and the Debug toolbar:



5. When stopped at a breakpoint, you can examine the values of variables. Locate the **Autos** window in Visual Studio and select the tab along the bottom named **Locals**. The **Locals** window shows local variables at the current point in the program. If you're stopped on the breakpoint set earlier, you see that the `pop` variable isn't yet defined. Now use the **Debug > Step Over** command (F10), and you see the value for `pop` appear:

| Name | Type |
|--------------|------------------|
| parent.env() | <environment> |
| mu | double (numeric) |
| N | double (numeric) |
| pop | double (numeric) |
| stddev | double (numeric) |

6. To examine variables in different scopes, including the global scope and package scopes, use the [Variable Explorer](#). The Variable Explorer also gives you the ability to switch to a tabular view with sortable columns and to export data to a CSV file.

| Variable Explorer | |
|-------------------|--|
| File | X |
| .GlobalEnv | Search |
| Name | Value |
| cars | 32 obs. of 11 variables |
| @.Data | List of 11 |
| @names | chr [1:11] "mpg" "cyl" "disp" "hp" "drat" "wt" "qs |
| @row.names | chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsu |
| [[1]] | "Mazda RX4" |
| [[2]] | "Mazda RX4 Wag" |
| [[3]] | "Datsun 710" |
| [[4]] | "Hornet 4 Drive" |
| [[5]] | "Hornet Sportabout" |
| [[6]] | "Valiant" |
| [[7]] | "Duster 360" |

7. You can continue stepping through the program line by line, or select **Continue** (F5) to run it to completion (or the next breakpoint).

To go deeper, see [Debugging](#) and [Variable Explorer](#).

Next steps

In this walkthrough you've learned the basics of R projects, using the interactive window, code editing, and debugging in Visual Studio. To continue exploring more capabilities, see the following articles as well as articles shown in the table of contents:

- [Sample projects](#)
- [Editing code](#)
- [Debugging](#)
- [Workspaces](#)
- [Visualizing data](#)

Tutorial: Create a Node.js and Express app in Visual Studio

3/21/2018 • 5 min to read • [Edit Online](#)

In this tutorial for Visual Studio development using Node.js and Express, you create a simple Node.js web application, add some code, explore some features of the IDE, and run the app. If you haven't already installed Visual Studio, install it for free [here](#).

In this tutorial, you learn how to:

- Create a Node.js project
- Add some code
- Use IntelliSense
- Run the app
- Hit a breakpoint

Prerequisites

- You must have Visual Studio installed and the Node.js development workload.

If you haven't already installed Visual Studio, install it for free [here](#).

If you need to install the workload but already have Visual Studio, click the **Open Visual Studio Installer** link in the left pane of the **New Project** dialog box. The Visual Studio Installer launches. Choose the **Node.js development** workload, then choose **Modify**.

- You must have the Node.js runtime installed.

If you don't have it installed, install the LTS version from the [Node.js](#) website. In general, Visual Studio automatically detects the installed Node.js runtime. If it does not detect an installed runtime, you can configure your project to reference the installed runtime in the properties page (after you create a project, right-click the project node and choose **Properties**).

This tutorial was tested with Node.js 8.10.0.

Create a project

First, you'll create an Node.js web application project.

1. Open Visual Studio 2017.
2. From the top menu bar, choose **File > New > Project...**
3. In the **New Project** dialog box, in the left pane, expand **JavaScript**, and then choose **Node.js**. In the middle pane, select **Basic Azure Node.js Express 4 Application**, and then choose **OK**.

If you don't see the **Basic Azure Node.js Express 4 Application** project template, you must install the **Node.js development** workload first.

Visual Studio creates the new solution and opens your project. The *app.js* project file opens in the editor (left pane).

- Highlighted in bold is your project, using the name you gave in the **New Project** dialog box. In the

file system, this project is represented by a `.njsproj` file in your project folder. You can set properties and environment variables associated with the project by right-clicking the project and choosing **Properties**. You can do round-tripping with other development tools, because the project file does not make custom changes to the Node.js project source.

- At the top level is a solution, which by default has the same name as your project. A solution, represented by a `.sln` file on disk, is a container for one or more related projects.
- The npm node shows any installed npm packages. You can right-click the npm node to search for and install npm packages using a dialog box.
- Project files such as `app.js` show up under the project node. `app.js` is the project startup file.

4. Open the **npm** node and make sure that all the required npm packages are present.

If any are missing (exclamation point icon), you can right-click the **npm** node and choose **Install Missing npm Packages**.

Add some code

1. In Solution Explorer (right pane), open the views folder, then open `index.pug`.
2. Replace the content with the following markup.

```
extends layout

block content
  h1= title
  p Welcome to #{title}
  script.
    var f1 = function() { document.getElementById('myImage').src='#{data.item1}' }
  script.
    var f2 = function() { document.getElementById('myImage').src='#{data.item2}' }
  script.
    var f3 = function() { document.getElementById('myImage').src='#{data.item3}' }

  button(onclick='f1()') One!
  button(onclick='f2()') Two!
  button(onclick='f3()') Three!
  p
  a: img(id='myImage' height='200' width='200' src='')
```

3. In the routes folder, open `index.js`.
4. Add the following code before the call to `router.get`:

```
var getData = function () {
  var data = {
    'item1': 'http://public-domain-photos.com/free-stock-photos-1/flowers/cactus-76.jpg',
    'item2': 'http://public-domain-photos.com/free-stock-photos-1/flowers/cactus-77.jpg',
    'item3': 'http://public-domain-photos.com/free-stock-photos-1/flowers/cactus-78.jpg'
  }
  return data;
}
```

5. Replace the `router.get` function call with the following code:

```

router.get('/', function (req, res) {
  res.render('index', { title: 'Express', "data" });
});

```

There is an error in the line of code containing `res.render`. We need to fix it before the app can run. We fix the error in the next section.

Use IntelliSense

1. In `index.js`, go to the line of code containing `res.render`.
2. After the `data` string, type `: get` and IntelliSense will show you the `getData` function. Select `getData`.

The screenshot shows a portion of the `index.js` file. Line 16 contains the code `res.render('index', { title: 'Express', "data": get })`. A cursor is positioned after the word `get`. An IntelliSense dropdown menu is open, listing several options, with `getData` highlighted. The dropdown also shows a preview of the `getData` function signature: `var getData: () => { [x: string]: any; 'item1': string; 'item2': string; 'item3': string; }`.

```

4
5   var getData = function () {
6     var data = {
7       'item1': 'http://public-domain-photos.com/free-stock-photos-1/flowers/cactus-76.jpg',
8       'item2': 'http://public-domain-photos.com/free-stock-photos-1/flowers/cactus-77.jpg',
9       'item3': 'http://public-domain-photos.com/free-stock-photos-1/flowers/cactus-78.jpg'
10    }
11    return data;
12  }
13
14  /* GET home page. */
15  router.get('/', function (req, res) {
16    res.render('index', { title: 'Express', "data": get });
17  });
18
19  module.exports = router;
20

```

3. Remove the comma (,) before `"data"` and you see green syntax highlighting on the expression. Hover over the syntax highlighting.

The screenshot shows the same `index.js` file. The line `res.render('index', { title: 'Express' "data": getData() })` now has green syntax highlighting on the word `data`. A tooltip appears over the word `data`, displaying its definition: `(property) "data": { [x: string]: any; 'item1': string; 'item2': string; 'item3': string; }`. Below the tooltip, an error message is shown: `(JS) ;' expected.`

```

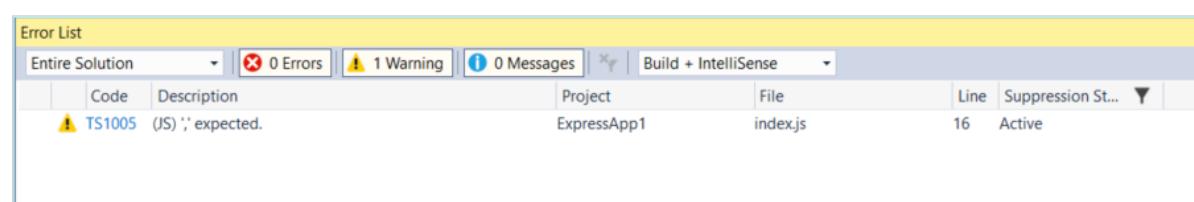
14  /* GET home page. */
15  router.get('/', function (req, res) {
16    res.render('index', { title: 'Express' "data": getData() });
17  });
18
19  module.exports = router;
20

```

The last line of this message tells you that the JavaScript interpreter expected a comma (,).

4. Click the **Error List** tab.

You see the warning and description along with the filename and line number.



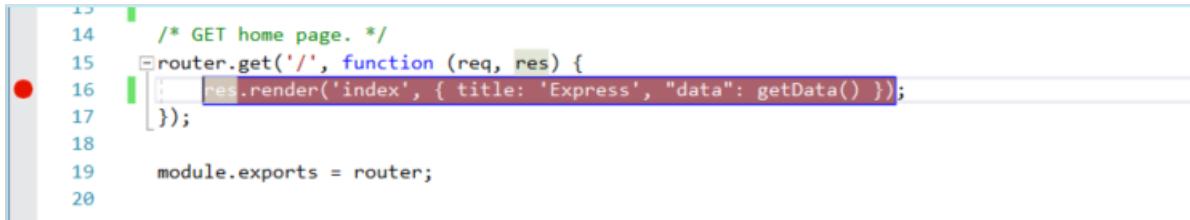
5. Fix the code by adding the comma (,) before "data".

Set a breakpoint

1. In *index.js*, click in the left gutter before the following line of code to set a breakpoint:

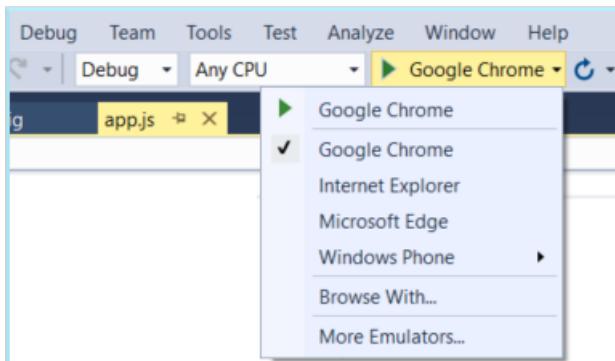
```
res.render('index', { title: 'Express', "data": getData() });
```

Breakpoints are the most basic and essential feature of reliable debugging. A breakpoint indicates where Visual Studio should suspend your running code so you can take a look at the values of variables, or the behavior of memory, or whether or not a branch of code is getting run.



Run the application

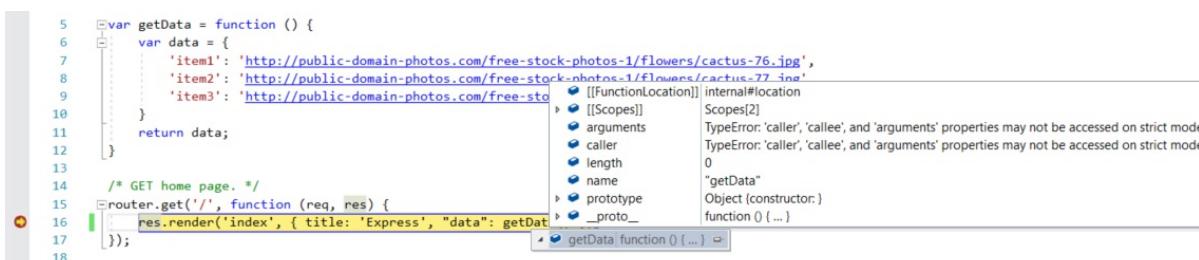
1. Select the debug target in the Debug toolbar.



2. Press **F5 (Debug > Start Debugging)** to run the application.

The debugger pauses at the breakpoint you set. Now, you can inspect your app state.

3. Hover over `getData` to see its properties in a DataTip



4. Press **F5 (Debug > Continue)** to continue.

The app opens in a browser.

In the browser window, you will see "Express" as the title and "Welcome to Express" in the first paragraph.

5. Click the buttons to display different images.

Express

Welcome to Express

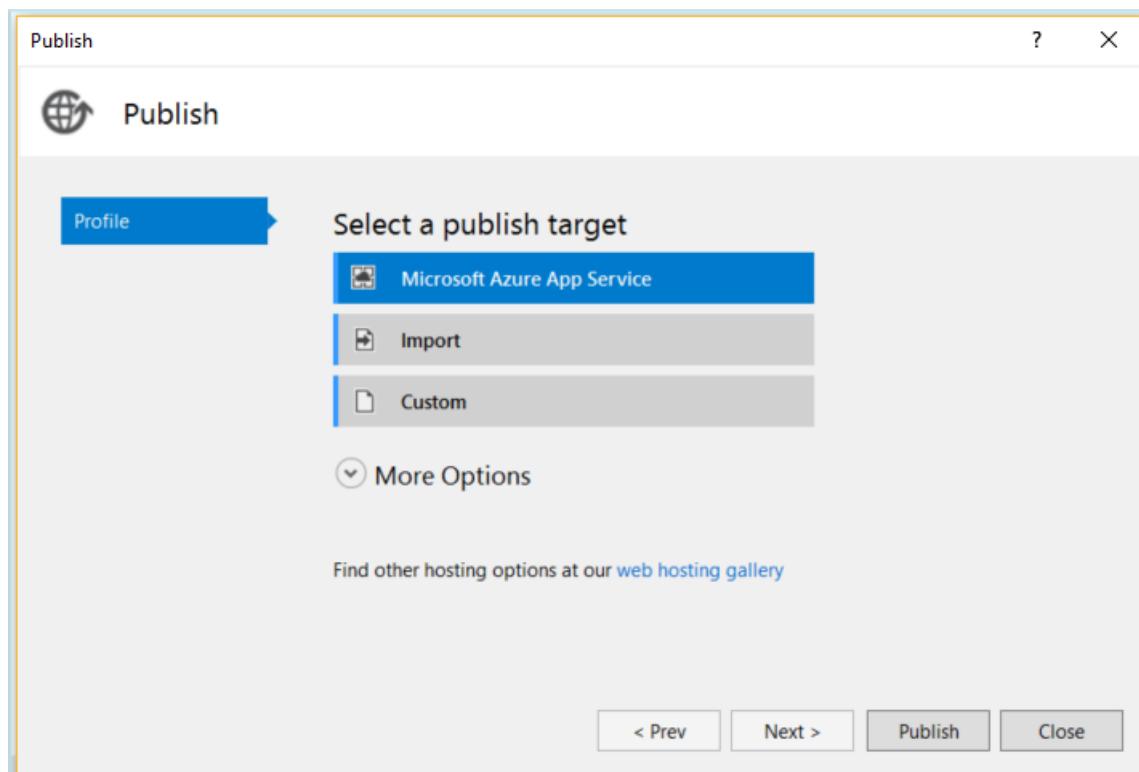
One! Two! Three!



6. Close the web browser.

(Optional) Publish to Azure App Service

1. In Solution Explorer, right-click the project and choose **Publish**.

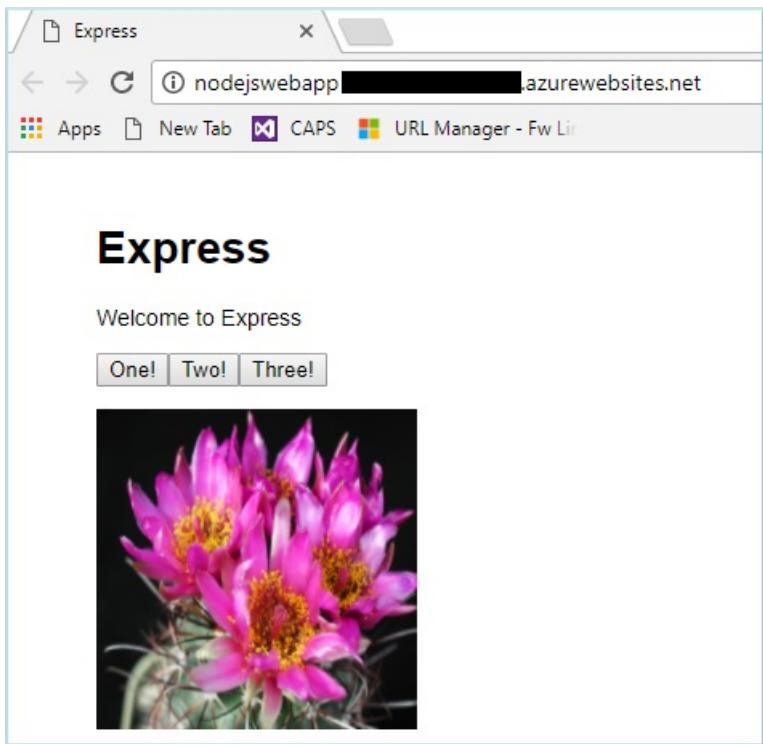


2. Choose **Microsoft Azure App Service**.

In the **App Service** dialog box, you can sign into your Azure account and connect to existing Azure subscriptions.

3. Follow the remaining steps to select a subscription, choose or create a resource group, choose or create an app service plan, and then follow the steps when prompted to publish to Azure. For more detailed instructions, see [Publish to Azure Website using Web Deploy](#).
4. The **Output** window shows progress on deploying to Azure.

On successful deployment, your app opens in a browser running in Azure App Service. Click a button to display an image.



Congratulations on completing this tutorial!

Next steps

In this tutorial, you learned how to create and run a Node.js app using Express and hit a breakpoint using the debugger.

[Node.js Tools for Visual Studio](#)

Getting started with C# and ASP.NET in Visual Studio

3/15/2018 • 5 min to read • [Edit Online](#)

In this tutorial for C# development with ASP.NET Core using Visual Studio, you'll create a C# ASP.NET Core web app, add code to it, explore some features of the IDE, and run the app.

If you haven't already installed Visual Studio, go to the [Visual Studio downloads](#) page to install it for free.

Before you begin

Here's a quick FAQ to introduce you to some key concepts.

What is C#?

C# is a type-safe and object-oriented programming language that's designed to be both robust and easy to learn.

What is ASP.NET Core?

ASP.NET Core is an open-source and cross-platform framework for building internet-connected applications, such as web apps and services. ASP.NET Core apps can run on either .NET Core or the .NET Framework. You can develop and run your ASP.NET Core apps cross-platform on Windows, Mac, and Linux. ASP.NET Core is open source at [GitHub](#).

What is Visual Studio?

Visual Studio is an integrated development suite of productivity tools for developers. Think of it as a program you can use to create programs and applications.

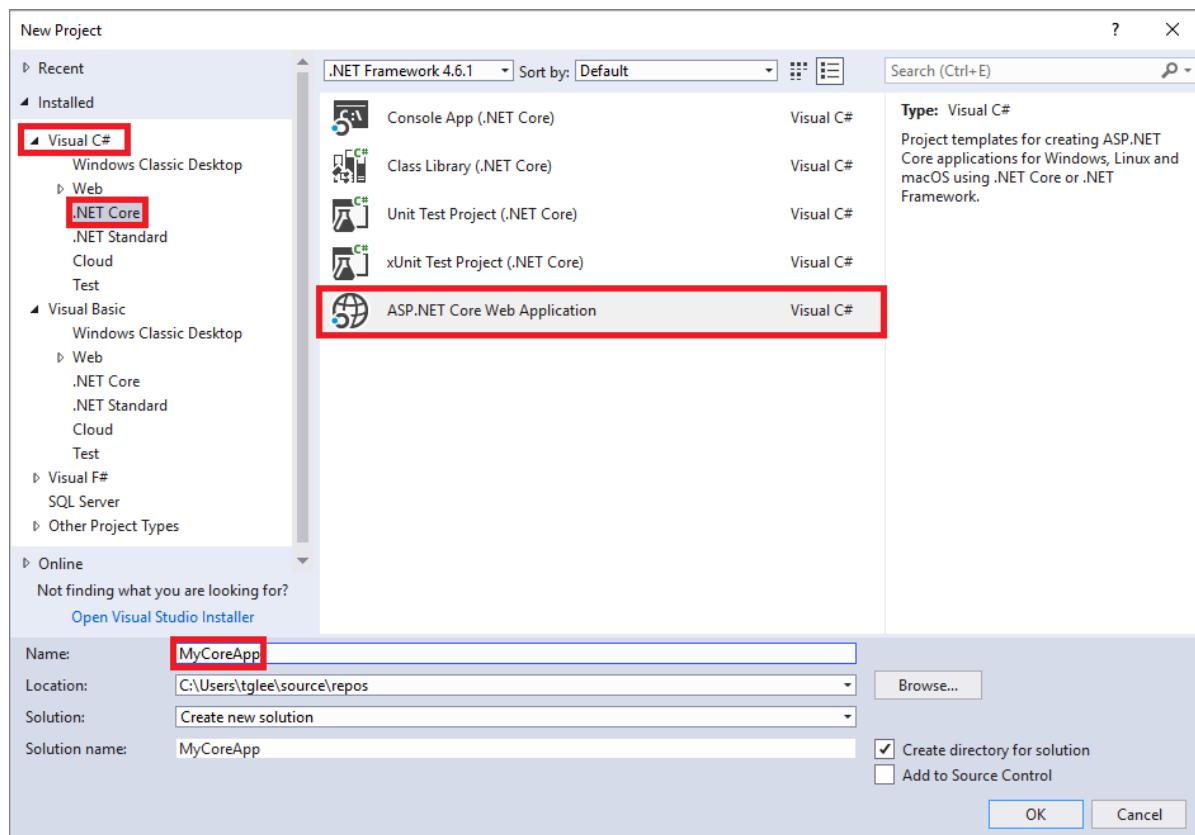
Start developing

Ready to start developing? Let's go!

Create a project

First, you'll create a ASP.NET Core project. The project type comes with all the template files you'll need, before you've even added anything.

1. Open Visual Studio 2017.
2. From the top menu bar, choose **File > New > Project....**
3. In the **New Project** dialog box in the left pane, expand **Visual C#**, expand **Web**, and then choose **.NET Core**. In the middle pane, choose **ASP.NET Core Web Application**, name the file *MyCoreApp*, and then choose **OK**.

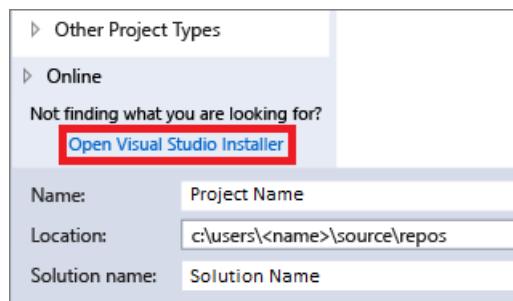


Add a workload (optional)

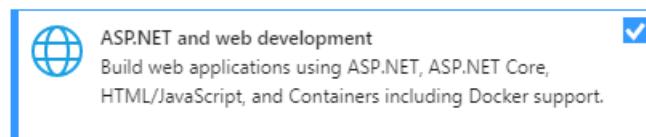
If you don't see the **ASP.NET Core Web Application** project template, you can get it by adding the **ASP.NET and web development** workload. You can add this workload in one of the two following ways, depending on which Visual Studio 2017 updates are installed on your machine.

Option 1: Use the **New Project** dialog box

1. Click the **Open Visual Studio Installer** link in the left pane of the **New Project** dialog box.



2. The Visual Studio Installer launches. Choose the **ASP.NET and web development** workload, and then choose **Modify**.



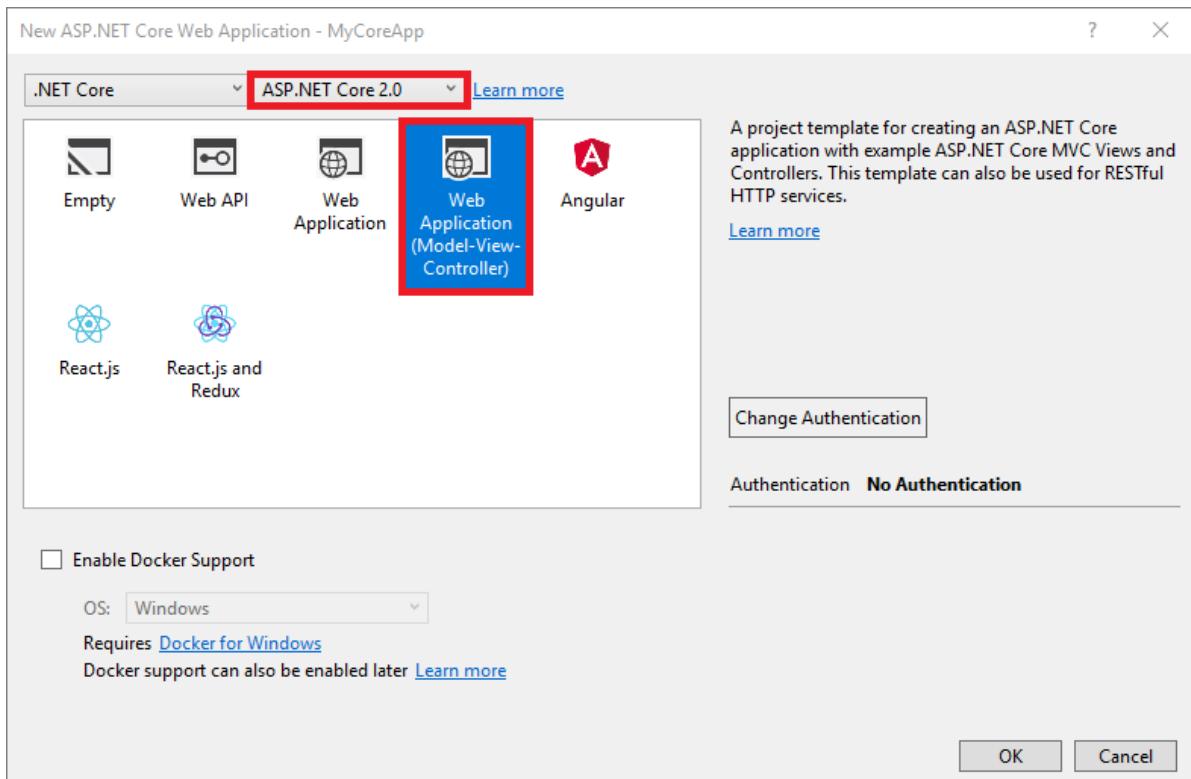
Option 2: Use the **Tools** menu bar

1. Cancel out of the **New Project** dialog box and from the top menu bar, choose **Tools > Get Tools and Features...**
2. The Visual Studio Installer launches. Choose the **ASP.NET and web development** workload, and then choose **Modify**.

Add a project template

1. In the **New ASP.NET Core Web Application** dialog box, choose the **Web Application (Model-View-Controller)** project template.

2. Select **ASP.NET Core 2.0** from the top drop-down menu. (If you don't see **ASP.NET Core 2.0** in the list, install it by following the **Download** link that should appear in a yellow bar near the top of the dialog box.) Choose **OK**.



About your solution

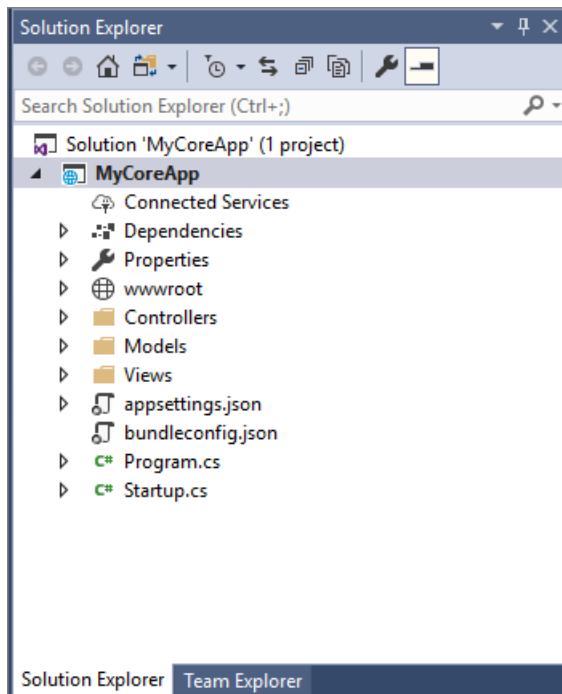
This solution follows the Model-View-Controller (MVC) architectural pattern that separates an app into three main components:

- **Models** include classes that represent the data of the app. The model classes use validation logic to enforce business rules for that data. Typically, model objects retrieve and store model state in a database.
- **Views** are the components that display the app's user interface (UI). Generally, this UI displays the model data.
- **Controllers** include classes that handle browser requests. They retrieve model data and call view templates that return a response. In an MVC app, the view displays only the information; the controller handles and responds to user input and interaction.

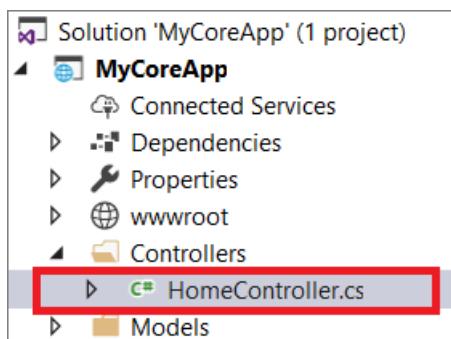
The MVC pattern helps you create apps that are easier to test and update than traditional monolithic apps.

Tour your solution

1. The project template creates a solution with a single ASP.NET Core project that is named **MyCoreApp**. Expand the project node to expose its contents.



2. Open the **HomeController.cs** file from the **Controllers** folder.

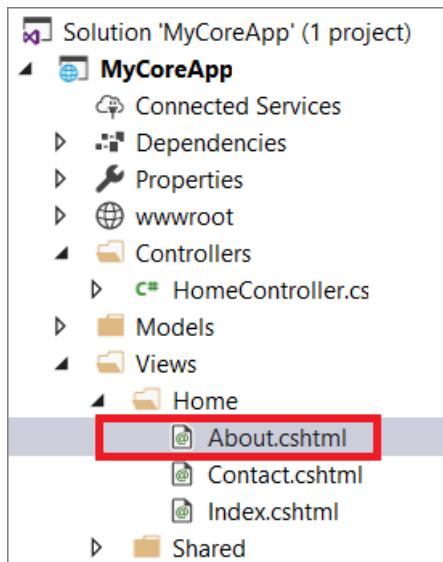


3. View the **HomeController.cs**

```
MyCoreApp
MyCoreApp.Controllers.HomeController
Index()

11  public class HomeController : Controller
12  {
13      public IActionResult Index()
14      {
15          return View();
16      }
17
18      public IActionResult About()
19      {
20          ViewData["Message"] = "Your application description page.";
21
22          return View();
23      }
}
```

4. The project also has a **Views** folder that contains other folders that map to each controller (as well as one for **Shared** views). For example, the view CSHTML file (an extension of HTML) for the **/Home/About** path would be at **Views/Home/About.cshtml**. Open that file.



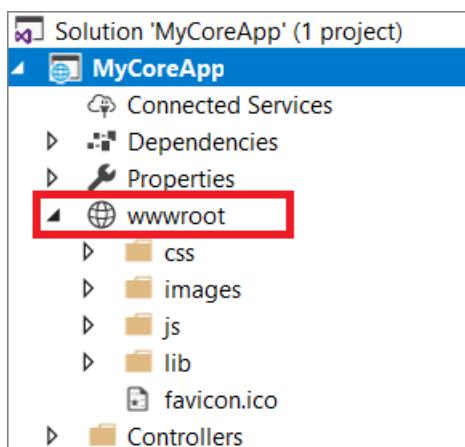
5. This CSHTML file uses the Razor syntax to render HTML based on a combination of standard tags and inline C#.

```
@{  
    ViewData["Title"] = "About";  
}  
<h2>@ViewData["Title"]</h2>  
<h3>@ViewData["Message"]</h3>  
  
<p>Use this area to provide additional information.</p>
```

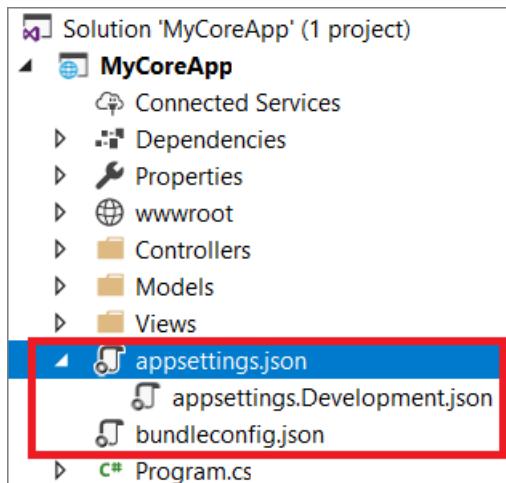
NOTE

To learn more about this, see the [Getting started with C# and ASP.NET using the Razor syntax page](#).

6. The solution also contains a **wwwroot** folder that is the root for your web site. You can put static site content, such as CSS, images, and JavaScript libraries, directly at the paths you'd want them to be at when the site is deployed.

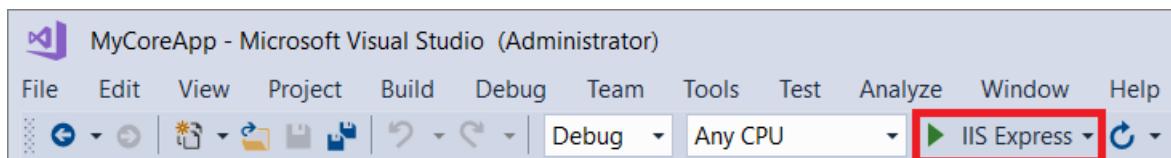


7. There are also a variety of configuration files that serve to manage the project, its packages, and the application at runtime. For example, the default application [configuration](#) is stored in **appsettings.json**. However, you can override some/all of these settings on a per-environment basis, such as by providing an **appsettings.Development.json** file for the **Development** environment.



Run and debug the application

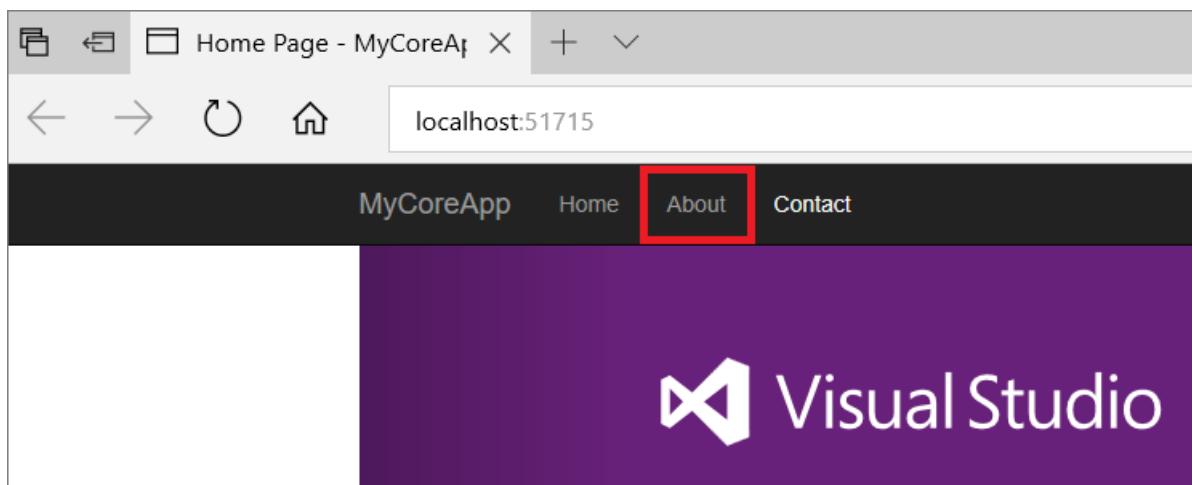
1. Choose the **IIS Express** button in the IDE to build and run the app in Debug mode. (Alternatively, press **F5**, or choose **Debug > Start Debugging** from the menu bar.)



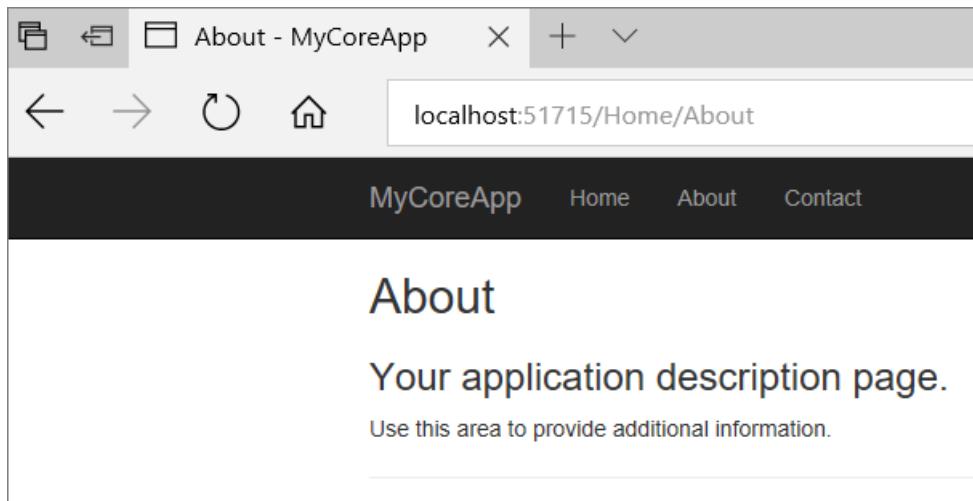
NOTE

If you get an error message that says **Unable to connect to web server 'IIS Express'**, close Visual Studio and then open it by using the **Run as administrator** option from the right-click or context menu. Then, run the application again.

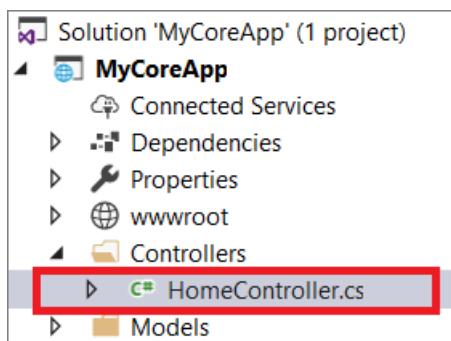
2. Visual Studio launches a browser window. Select **About**.



Among other things, the About page in the browser renders the text that is set in the `HomeController.cs` file.



3. Keep the browser window open and return to Visual Studio. Open **Controllers/HomeController.cs** if it's not already open.



4. Set a breakpoint in the first line of the **About** method. To do this, click in the margin or set the cursor on the line and press **F9**.

This line sets some data in the **ViewData** collection that is rendered in the CSHTML page at **Views/Home/About.cshtml**.

```
HomeController.cs   X   Startup.cs
MyCoreApp   MyCoreApp.Controllers.HomeController   About()
7   using MyCoreApplication.Models;
8
9   namespace MyCoreApplication.Controllers
10 {
11       public class HomeController : Controller
12    {
13        public IActionResult Index()
14        {
15            return View();
16        }
17
18        public IActionResult About()
19        {
20            ViewData["Message"] = "Your application description page.";
21
22            return View();
23        }
24}
```

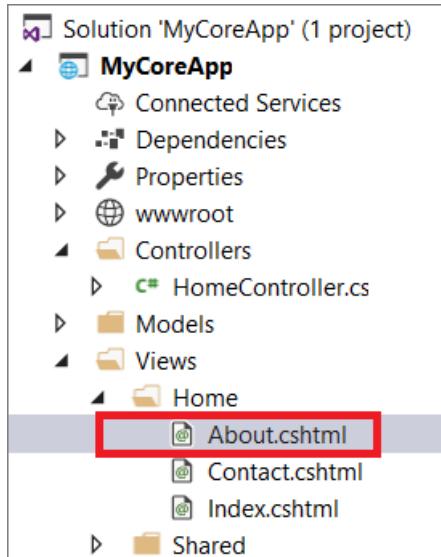
The screenshot shows the Visual Studio code editor with the "HomeController.cs" file open. The code defines a controller with two actions: "Index" and "About". The "About" action sets the "Message" key in the ViewData collection to the string "Your application description page.". A red rectangular selection box highlights the line of code "ViewData["Message"] = "Your application description page.;"".

5. Return to the browser and refresh the About page. This will trigger the breakpoint in Visual Studio.
6. In Visual Studio, mouse over the **ViewData** member to view its data.



7. Remove the application breakpoint using the same method you used to add it.

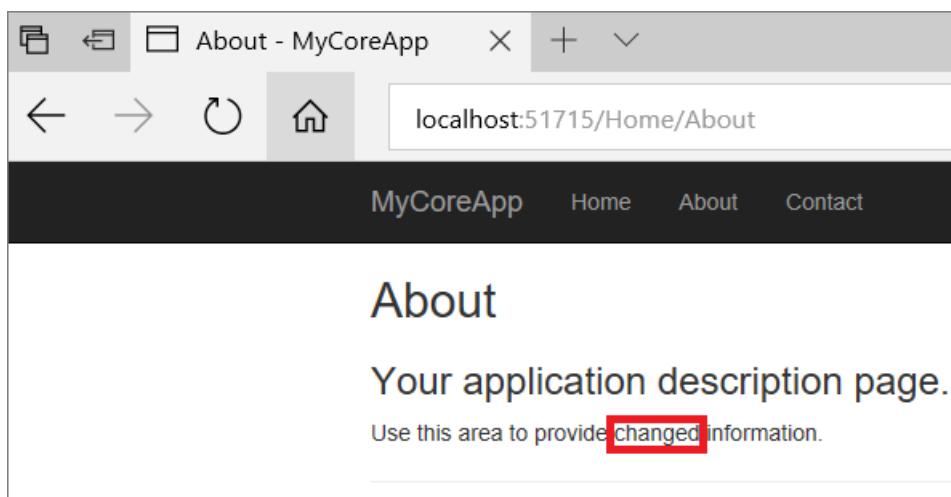
8. Open **Views/Home/About.cshtml**.



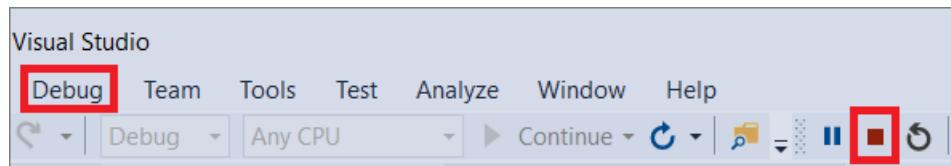
9. Change the text "**additional**" to "**changed**" and save the file.

```
1  @{
2      ....ViewData["Title"] = "About";
3  }
4  <h2>@ViewData["Title"]</h2>
5  <h3>@ViewData["Message"]</h3>
6
7  <p>Use this area to provide changed information.</p>
```

10. Return to the browser window to see the updated text. (Refresh the browser if you don't see the text that you changed.)



11. Choose the **Stop Debugging** button from the toolbar to stop debugging. (Alternatively, press **Shift+F5**, or choose **Debug > Stop Debugging** from the menu bar.)



Next steps

Congratulations on completing this tutorial! We hope you learned a little bit about C#, ASP.NET Core, and the Visual Studio IDE. To learn even more, continue with the following tutorial.

[Getting started with ASP.NET Core MVC and Visual Studio](#)

Getting started with Visual Basic in Visual Studio

3/15/2018 • 3 min to read • [Edit Online](#)

In this tutorial for Visual Basic (VB), you'll use Visual Studio to create and run a few different console apps, and explore some features of the Visual Studio [integrated development environment \(IDE\)](#) while you do so.

If you haven't already installed Visual Studio, go to the [Visual Studio downloads](#) page to install it for free.

Before you begin

Here's a quick FAQ to introduce you to some key concepts.

What is Visual Basic?

Visual Basic is a type-safe programming language that's designed to be easy to learn. It is derived from BASIC, which means "Beginner's All-purpose Symbolic Instruction Code".

What is Visual Studio?

Visual Studio is an integrated development suite of productivity tools for developers. Think of it as a program you can use to create programs and applications.

What is a console app?

A console app takes input and displays output in a command-line window, a.k.a. a console.

What is .NET Core?

.NET Core is the evolutionary next step of the .NET Framework. Where the .NET Framework allowed you to share code across programming languages, .NET Core adds the ability to share code across platforms. Even better, it's open source. (Both the .NET Framework and .NET Core include libraries of prebuilt functionality as well as a common language runtime (CLR), which acts as a virtual machine in which to run your code.)

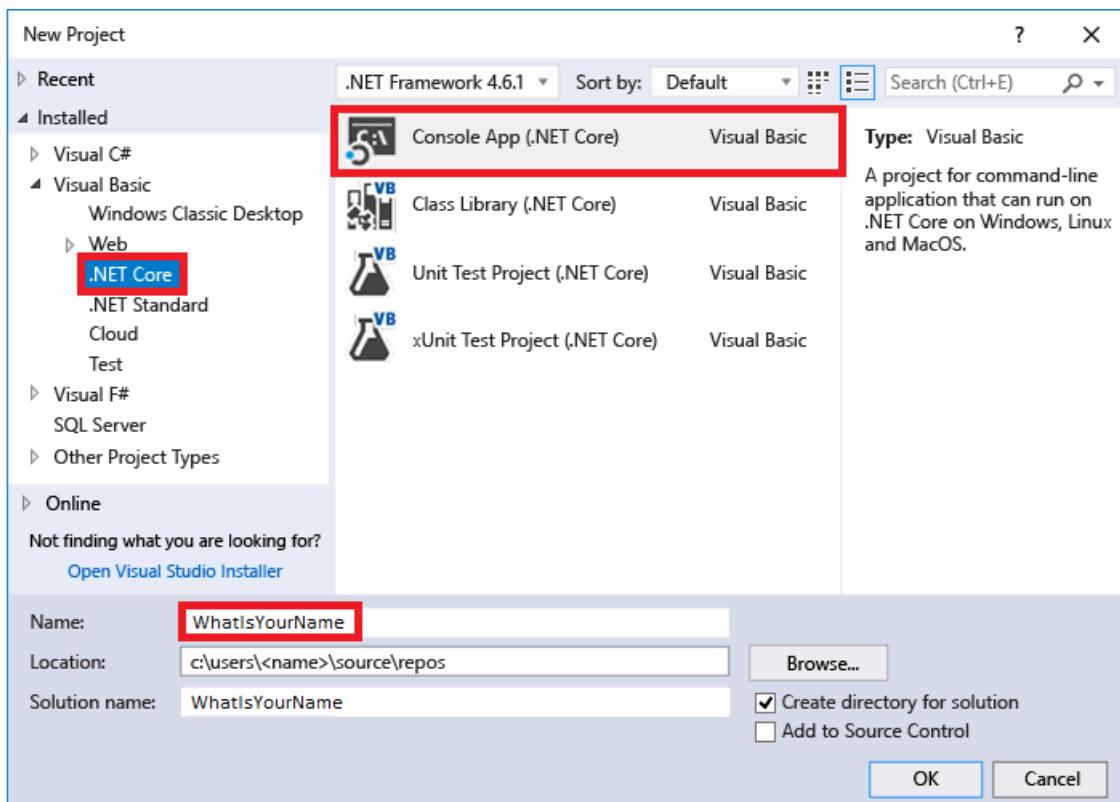
Start developing

Ready to start developing? Let's go!

Create a project

First, we'll create a Visual Basic application project. The project type comes with all the template files you'll need, before you've even added anything!

1. Open Visual Studio 2017.
2. From the top menu bar, choose **File > New > Project...**.
3. In the **New Project** dialog box in the left pane, expand **Visual Basic**, and then choose **.NET Core**. In the middle pane, choose **Console App (.NET Core)**. Then name the file *HelloWorld*.

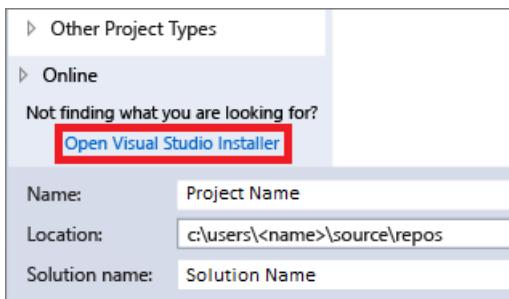


Add a workgroup (optional)

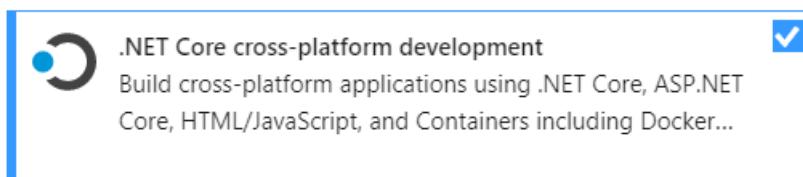
If you don't see the **Console App (.NET Core)** project template, you can get it by adding the **.NET Core cross-platform development** workload. You can add this workload in one of the two following ways, depending on which Visual Studio 2017 updates are installed on your machine.

Option 1: Use the New Project dialog box

1. Click the **Open Visual Studio Installer** link in the left pane of the **New Project** dialog box.



2. The Visual Studio Installer launches. Choose the **.NET Core cross-platform development** workload, and then choose **Modify**.



Option 2: Use the Tools menu bar

1. Cancel out of the **New Project** dialog box and from the top menu bar, choose **Tools > Get Tools and Features...**
2. The Visual Studio Installer launches. Choose the **.NET Core cross-platform development** workload, and then choose **Modify**.

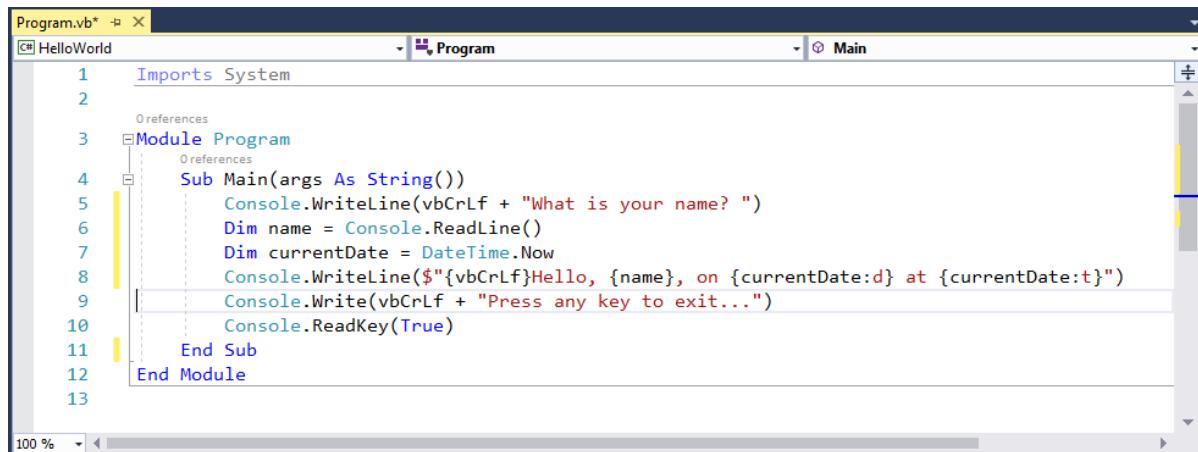
Create a "What Is Your Name" application

Let's create an app that prompts you for your name and then displays it along with the date and time. Here's how:

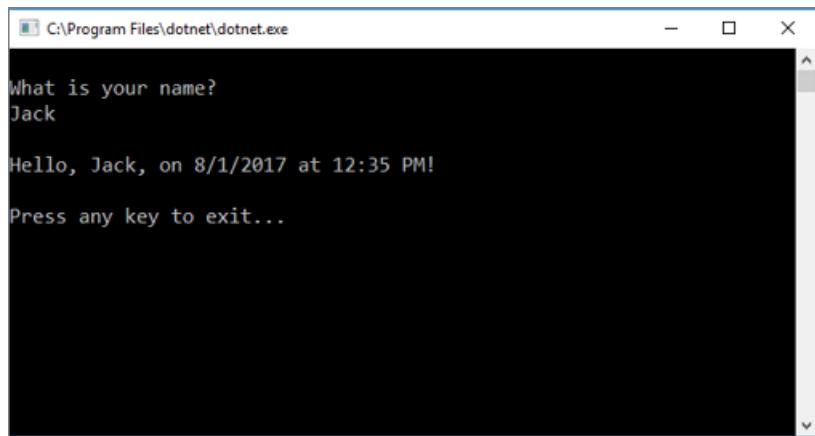
1. If it is not already open, then open your *WhatIsYourName* project.
2. Enter the following Visual Basic code immediately after the opening bracket that follows the `Sub Main(args As String())` line and before the `End Sub` line:

```
Console.WriteLine(vbCrLf + "What is your name? ")
Dim name = Console.ReadLine()
Dim currentDate = DateTime.Now
Console.WriteLine($"{vbCrLf}Hello, {name}, on {currentDate:d} at {currentDate:t}")
Console.Write(vbCrLf + "Press any key to exit... ")
Console.ReadKey(True)
```

This code replaces the existing `WriteLine`, `Write`, and `ReadKey` statements.



3. When the console window opens, enter your name. Your console window should look similar to the following screenshot:



4. Press any key to close the console window.

Create a "Calculate This" application

1. Open Visual Studio 2017, and then from the top menu bar, choose **File > New > Project....**
2. In the **New Project** dialog box in the left pane, expand **Visual Basic**, and then choose **.NET Core**. In the middle pane, choose **Console App (.NET Core)**. Then name the file *CalculateThis*.
3. Enter the following code between the `Module Program` line and `End Module` line:

```

Public num1 As Integer
Public num2 As Integer
Public answer As Integer
Sub Main()
    Console.WriteLine("Type a number and press Enter")
    num1 = Console.ReadLine()
    Console.WriteLine("Type another number to add to it and press Enter")
    num2 = Console.ReadLine()
    answer = num1 + num2
    Console.WriteLine("The answer is " & answer)
    Console.ReadLine()
End Sub

```

Your code window should look like the following screenshot:

```

Program.vb* ✎ X
VB CalculateThis Program num2
1 Imports System
2
3 Module Program
4     Public num1 As Integer
5     Public num2 As Integer
6     Public answer As Integer
7     Sub Main()
8         Console.WriteLine("Type a number and press Enter")
9         num1 = Console.ReadLine()
10        Console.WriteLine("Type another number to add to it and press Enter")
11        num2 = Console.ReadLine()
12        answer = num1 + num2
13        Console.WriteLine("The answer is " & answer)
14        Console.ReadLine()
15    End Sub
16 End Module
17

```

- Click **CalculateThis** to run your program. Your console window should look similar to the following screenshot:

```

C:\Program Files\dotnet\dotnet.exe
Type a number and press Enter
5
Type another number to add to it and press Enter
5
The answer is 10

```

Next steps

Congratulations on completing this tutorial! To learn even more about Visual Basic and the Visual Studio IDE, see

the following pages.

- [Visual Basic Guide](#)
- [What's new in Visual Basic](#)
- [IntelliSense for Visual Basic code files](#)
- [Visual Basic Language Reference](#)
- [Visual Basic Fundamentals for Absolute Beginners](#) video course

Get started with debugging in Visual Studio

3/8/2018 • 9 min to read • [Edit Online](#)

Visual Studio provides a powerful integrated set of project build and debugging tools. In this topic, find out how to start using the most basic set of debugging UI features.

If you haven't already installed Visual Studio, go to the [Visual Studio Downloads](#) page to install it for free.

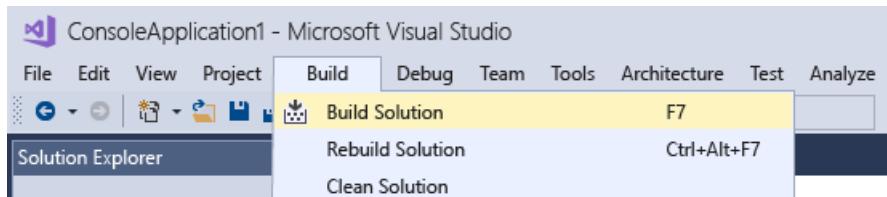
My code doesn't work. Help me, Visual Studio!

So you've figured out the editor and you've created some code. Now, you want to start debugging that code. In Visual Studio, as with most IDEs, there are two phases to debugging: building the code to catch and resolve project and compiler errors; and running that code in the environment to catch and resolve run-time and dynamic errors.

Build your code

There are two basic types of build configuration: **Debug** and **Release**. The first configuration produces a slower, larger executable that allows for a richer interactive run-time debugging experience, but should never be shipped. The second builds a faster, more optimized executable that's appropriate to ship (at least from the perspective of the compiler). The default build configuration is **Debug**.

The easiest way to build your project is to press **F7**, but you can also start the build by selecting **Build > Build Solution** from the main menu.



You can observe the build process in the **Output** status window at the bottom of the Visual Studio UI. Errors, warnings, and build operations are displayed here. If you have errors (or if you have a warnings above a configured level), your build will fail. You can click on the errors and warnings to go to the line where they occurred. Rebuild your project by pressing either **F7** again (to recompile only the files with errors) or **Ctrl+Alt+F7** (for a clean and complete rebuild).

There are two build tabbed windows in the results window below the editor: the **Output** window, which contains the raw compiler output (including error messages); and the **Error List** window, which provides a sortable and filterable list of all errors and warnings.

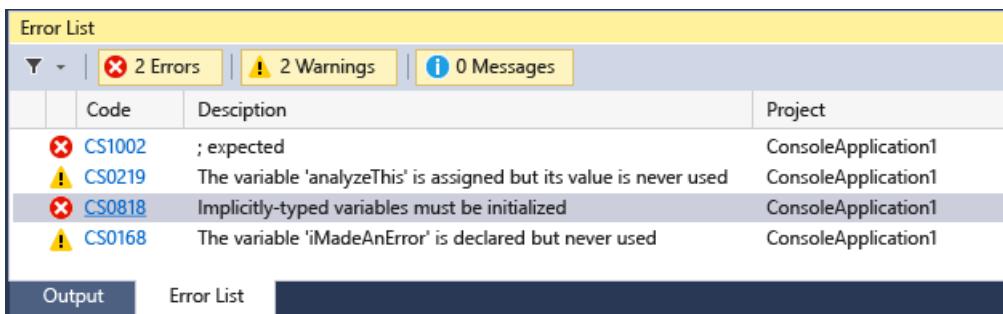
When successful, you will see results like this in the **Output** window.

A screenshot of the "Output" window in Visual Studio. The window has a yellow header bar with the word "Output". Below the header, there is a toolbar with icons for "Show output from:" dropdown (set to "Build"), copy, cut, paste, and other options. The main content area displays the build log: "1>----- Build started: Project: MyNewApp, Configuration: Release Any CPU -----", "1> MyNewApp -> c:\users\user1\documents\visual studio 2015\Projects\MyNewApp\MyNewApp\bin\Release\MyNewApp.exe", and "===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====".

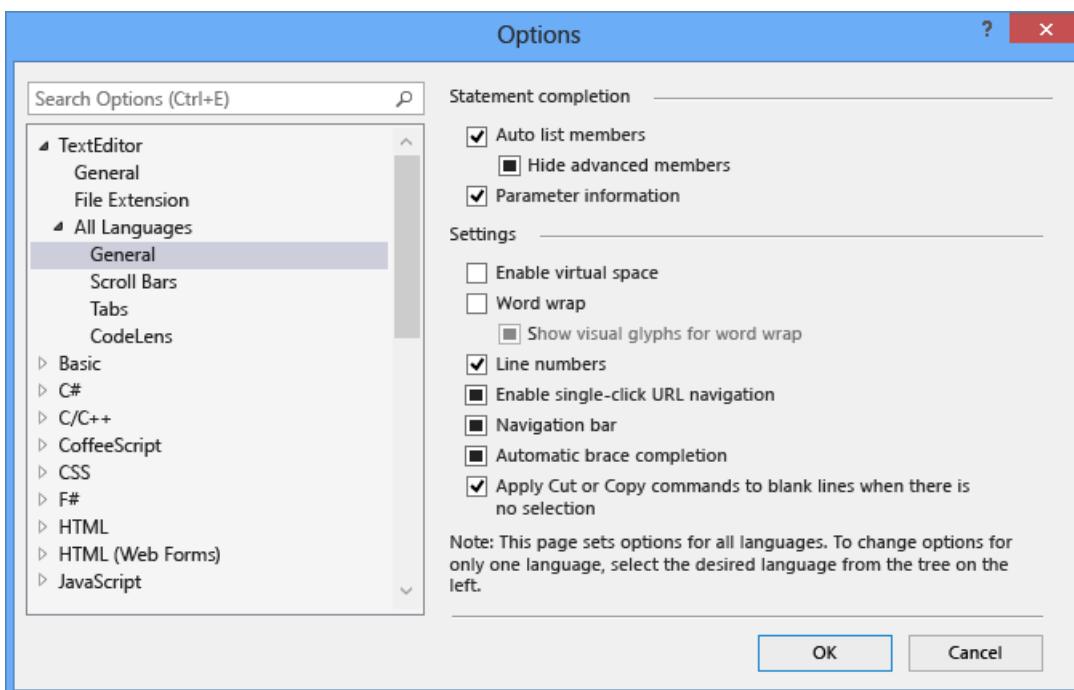
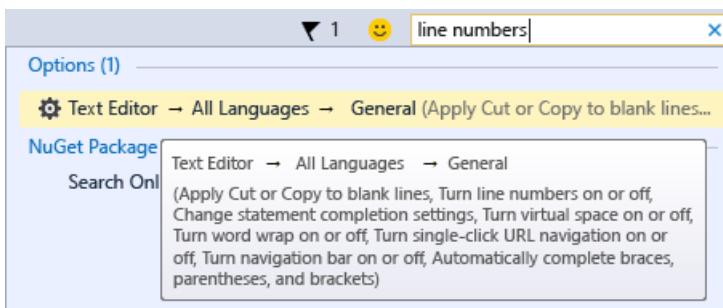
Review the Error List

Unless you've made no modifications to code you've previously and successfully compiled, you probably have an error. If you're new to coding, you probably have lots of them. Errors are sometimes obvious, such as a simple syntax error or incorrect variable name, and sometimes they are difficult to understand, with only a cryptic code to guide you. For a cleaner view of the issues, navigate to the bottom of the build **Output** window, and click the **Error**

List tab. This takes you to a more organized view of the errors and warnings for your project, and gives you some extra options as well.

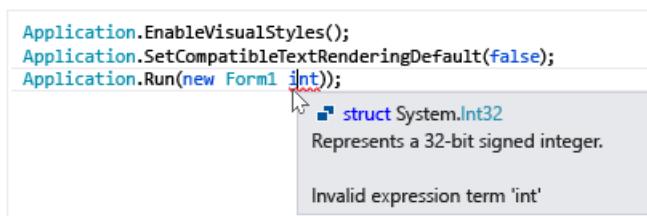


Click on the error line in the **Error List** window and jump to the line the error occurs in. (Or turn on line numbers by clicking in the **Quick Launch** bar in the upper-right, typing "line numbers" into it, and pressing Enter. This is the fastest way to get to **Options** window entry where you can turn on line numbers. Learn to use the **Quick Launch** bar and save yourself a lot of UI clicks!)

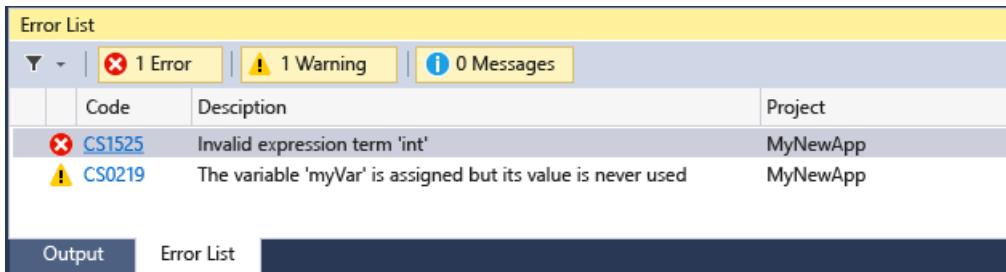


Use **Ctrl+G** to quickly jump to the line number where the error occurred.

The error is identified by a red "squiggle" underscore. Hover over it for additional details. Make the fix and it will go away, although you may introduce a new error with the correction. (This is called a "regression").

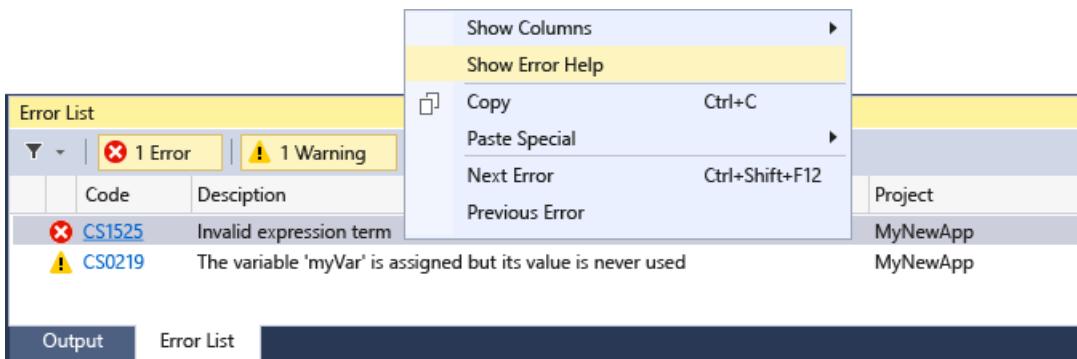


Walk through the error list and address all the errors in your code.



Review errors in detail

Many errors may make no sense to you, phrased as they are in the terms of the compiler. In those cases, you will need additional information. From the **Error List** window, you can do an automatic Bing search for more information on the error (or warning) by right-clicking on the corresponding entry line and selecting **Show Error Help** from the context menu.

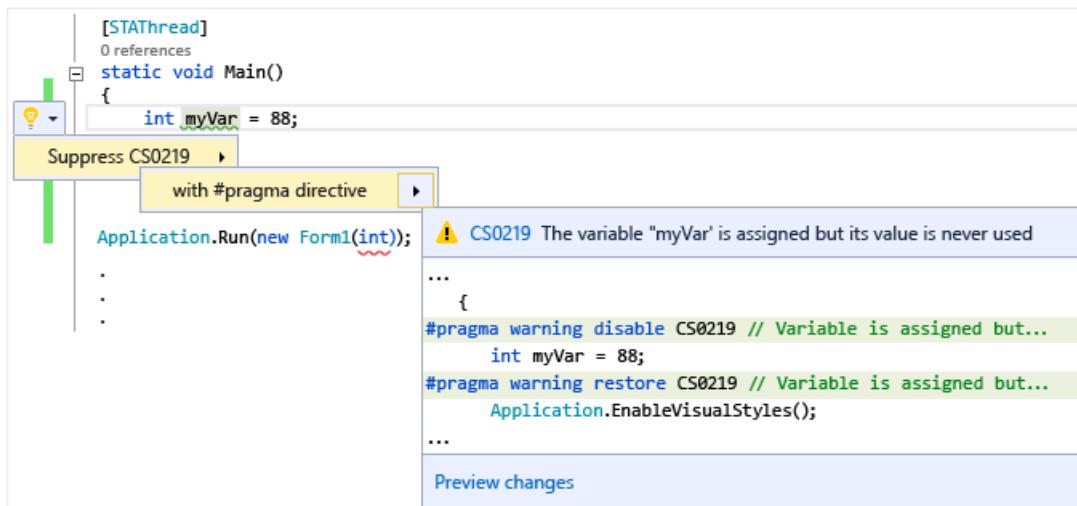


This launches a tab inside Visual Studio that hosts the results of a Bing search for the error code and text. The results are from many different sources on the Internet, and not all may be helpful.

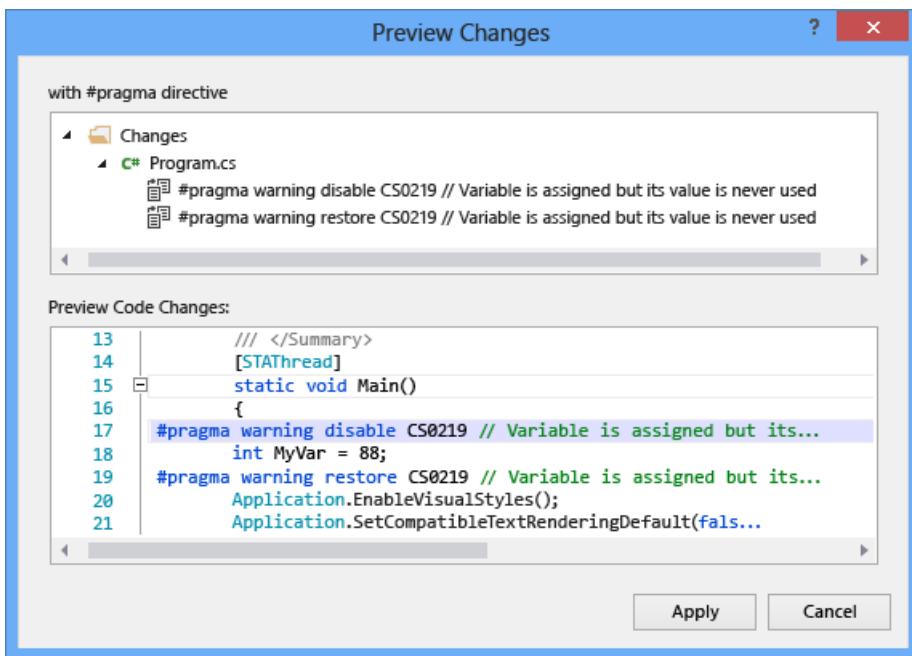
Alternatively, you can click on the hyperlinked error code value in the **Code** column of the **Error List**. This will launch a Bing search for just the error code.

Use Light Bulbs to fix or refactor code

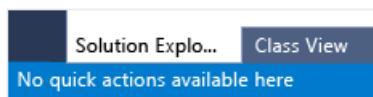
Light Bulbs are a new feature for Visual Studio that let you refactor code inline. They are an easy way to fix common warnings quickly and effectively. To access them, right-click on a warning squiggle (or press **Ctrl+.** while hovering over the squiggle), and then select **Quick Actions**.



You will see a list of possible fixes or refactors you can apply to that line of code.



Light Bulbs can be used wherever the code analyzers determine there is an opportunity to fix, refactor, or improve your code. Click on any line of code, right-click to open the context menu, and select **Quick Actions** (or, again, if you prefer efficiency, press **Ctrl+.**). If there are refactoring or improvement options available, they will be displayed; otherwise, the message **No quick options available here** will be displayed in the lower-left corner bezel of the IDE.

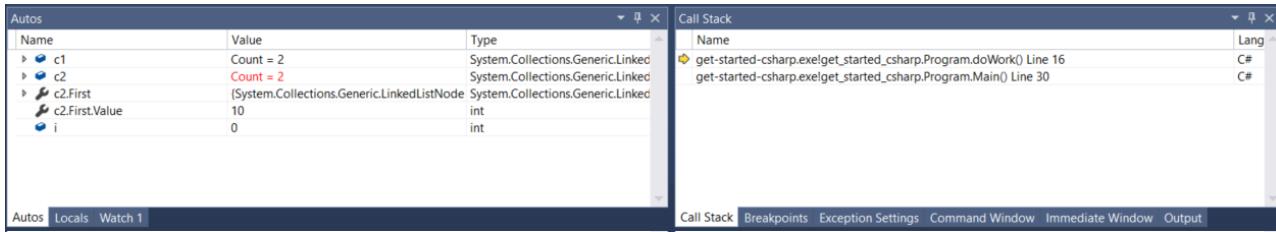


With experience, you can quickly use the arrow keys and **Ctrl+.** to check for Quick Option refactoring opportunities and clean up your code!

For more information on Light Bulbs, read [Perform quick actions with light bulbs](#).

Debug Your running code

Now that you've successfully built your code and performed a little clean up, run it by pressing **F5** or selecting **Debug > Start Debugging**. This will start your app in a debug environment so you can observe its behavior in detail. The Visual Studio IDE changes while your app is running: the **Output** window is replaced by two new ones (in the default window configuration), the **Autos/Locals/Watch** tabbed window and the **Call Stack/Breakpoints/Exception Settings/Output** tabbed window. These windows have multiple tabs which allow you to inspect and evaluate your app's variables, threads, call stacks, and various other behaviors as it runs.



You can stop your app by pressing **Shift+F5** or by clicking the **Stop** button. Or, you can simply close the app's main window (or command line dialog).

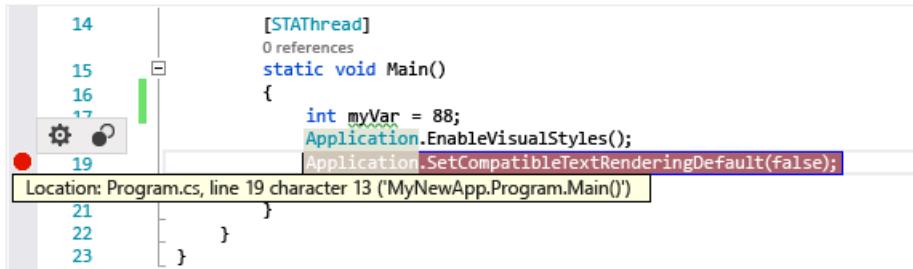
If your code ran perfectly and exactly as expected, congratulations! However, if it hung, or crashed, or gave you some strange results, you'll need to find the source of those problems and fix the bugs.

Set simple breakpoints

Breakpoints are the most basic and essential feature of reliable debugging. A breakpoint indicates where Visual Studio should suspend your running code so you can take a look at the values of variables, or the behavior of

memory, or whether or not a branch of code is getting run. You do NOT need to rebuild a project after setting and removing breakpoints.

Set a breakpoint by clicking in the far margin of the line where you want the break to occur, or press **F9** to set a breakpoint on the current line of code. When you run your code, it will pause (or *break*) before the instructions for this line of code are executed.



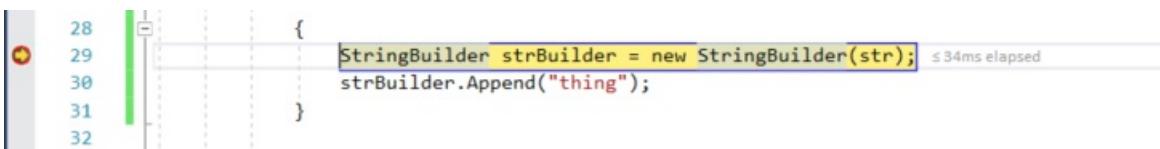
Common uses for breakpoints include:

1. To narrow down the source of a crash or hang, scatter them throughout and around the code of the method call you think is causing the failure. As you run code in the debugger, remove and then reset the breakpoints closer together until you find the offending line of code. See the next section to learn how to run code in the debugger.
2. When you introduce new code, set a breakpoint at the beginning of it and run the code to make sure it is behaving as expected.
3. If you have implemented a complicated behavior, set breakpoint(s) for the algorithmic code so you can inspect the values of the variables and data when the program breaks.
4. If you are writing C or C++ code, use breakpoints to stop the code so you can inspect address values (look for NULL) and reference counts when debugging for memory-related failures.

For more information on using breakpoints, read [Using Breakpoints](#).

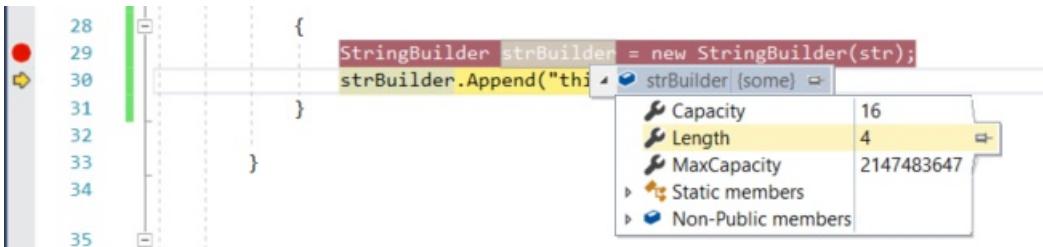
Inspect your code at run-time

When your running code hits a breakpoint and pauses, the line of code marked in yellow (the current statement) has not executed yet. At this point, you may want to execute the current statement and then inspect the changed values. You can use several *step* commands to execute code in the debugger. If the marked code is a method call, you can step into it by pressing **F11**. You can also *step over* the line of code by pressing **F10**. For additional commands and details on how to step through code, read [Navigate code with the debugger](#).



In the preceding illustration, you can advance the debugger one statement by pressing either **F10** or **F11** (since there is no method call here, both commands have the same result).

While the debugger is paused, you can inspect your variables and call stacks to determine what is going on. Are the values in the ranges you expect to see? Are calls being made in the right order?



Hover over a variable to see the value(s) and reference(s) it currently contains. If you see a value you didn't expect, you probably have a bug in the preceding or calling lines of code. For more in-depth information, [learn more](#) about using the debugger.

Additionally, Visual Studio displays the Diagnostic Tools window, where you can observe your app's CPU and memory usage over time. Later in your app development, you can use these tools to look for unanticipated heavy CPU usage or memory allocation. Use it in conjunction with the **Watch** window and breakpoints to determine what's causing unexpected heavy usage or unreleased resources. For more information, see [Profiling feature tour](#).

Run unit tests

Unit tests are your first line of defense against code bugs because, when done correctly, they test a single "unit" of code, typically a single function, and are usually much easier to debug than debugging your full program. Visual Studio installs the Microsoft unit testing frameworks for both 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 edition, you can run tests automatically after every build.

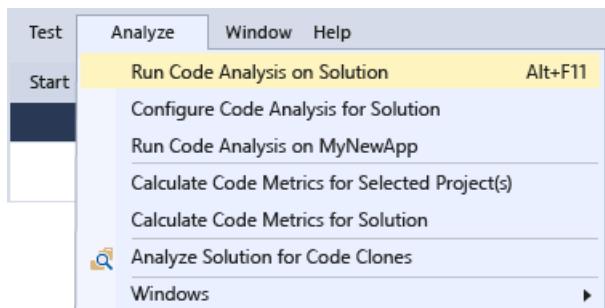
To get started, read [Generate unit tests for your code with IntelliTest](#).

To learn more about unit tests in Visual Studio and how they can help you create better quality code, read [Unit Test Basics](#).

Perform static code analysis

"Static code analysis" is a fancy way of saying "automatically check my code for common problems that can lead to run-time errors or problems in code management". Get in the habit of running it once you've cleaned up the obvious errors preventing build, and take some time to address the warnings it may produce. You'll save yourself some headaches down the road, as well as learn a few code style techniques.

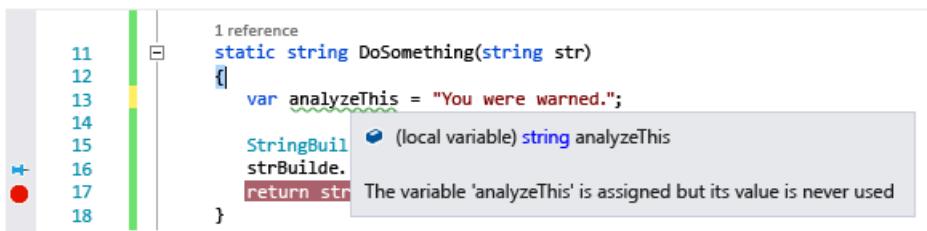
Press **Alt+F11** (or select **Analyze > Run Code Analysis on Solution** from the top menu) to start static code analysis. This may take some time if you have a lot of code.



Any new or updated warnings will appear in the **Error List** tab at the bottom of the IDE. Click on the warnings to jump to them.



The warnings will be identified with a bright yellow-green squiggle instead of a red one. Hover over them for more detail, and right-click on them to get a context menu to assist in fixes or refactoring options.



A screenshot of a debugger interface showing a code editor and a status bar. The code editor displays a C# method named `DoSomething`. The status bar at the bottom right shows a warning message: "The variable 'analyzeThis' is assigned but its value is never used".

```
1 reference
static string DoSomething(string str)
{
    var analyzeThis = "You were warned.";
    StringBuilder strBuilder;
    return str;
}
```

See Also

[Debugger Feature Tour](#)

[Learn more about using the debugger](#)

Learn how to use the code editor and other tools in Visual Studio to write, navigate, manage, and improve your code.

□ [Create solutions and projects](#)

□ [Use the code editor](#)

□ [Navigate code](#)

□ [Generate and fix code](#)

□ [Use Visual Studio without projects or solutions](#)

□ [Manage your code using Git with Visual Studio](#)

Get started with programming languages in Visual Studio

Create your first app in Visual Studio.

[Get started with C# and ASP.NET in Visual Studio](#)

[Get started with C++ in Visual Studio](#)

[Get started with Python in Visual Studio](#)

[Create a console app with Visual Basic](#)

[Create a web app with Node.js](#)

[Get started with R Tools for Visual Studio](#)

How to: Move Around in the Visual Studio IDE

12/22/2017 • 4 min to read • [Edit Online](#)

The integrated development environment (IDE) has been designed to allow you to move from window to window and file to file in several different ways, depending on your preference or project requirements. You can choose to cycle through open files in the editor, or cycle through all active tool windows in the IDE. You also can switch directly to any file open in the editor, regardless of the order in which it was last accessed. These features can help increase your productivity when working in the IDE.

NOTE

The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in this article, depending on your active settings or edition. This article was written with **General** settings in mind. To change your settings, for example to **General** or **Visual C++** settings, choose **Tools, Import and Export Settings**, and then choose **Reset all settings**.

Keyboard Shortcuts

Almost every menu command in Visual Studio has a keyboard shortcut. You can also create your own custom shortcuts. For more information, see [Identifying and Customizing Keyboard Shortcuts](#).

Navigating Among Files in the Editor

You can use several methods to move through the files open in the editor. You can move among files based on the order in which you access them, use the IDE Navigator to quickly find any file currently open, or pin favorite files to the tab well so that they are always visible.

Navigate Backward and Navigate Forward cycle through the open files in the editor based on the order in which they were accessed, much like Back and Forward do for your viewing history in Microsoft Internet Explorer.

To move through open files in order of use

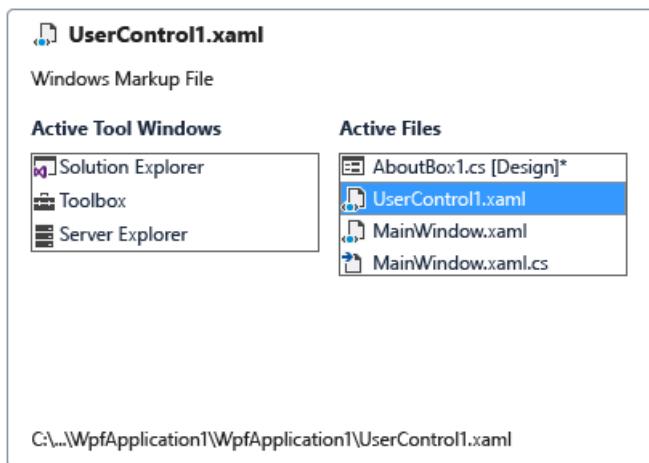
- To activate open documents in the order they were most recently touched, press **Ctrl+-**.
- To activate open documents in the reverse order, press **Ctrl+Shift+-**.

NOTE

Navigate Backward and **Navigate Forward** also can be found on the **View** menu.

You also can switch to a specific file open in the editor, regardless of when you last accessed the file, using the **IDE Navigator**, the **Active Files** list in the editor, or the **Windows** dialog box.

The **IDE Navigator** works much like the Windows application switcher. It is not available from menus and can be accessed only using shortcut keys. You can use either of two commands to access the **IDE Navigator** (shown below) to cycle through files, depending on the order in which you want to cycle through.



`Window.PreviousDocumentWindowNav` allows you to move to the file most recently accessed and `Window.NextDocumentWindowNav` allows you to move in the reverse order. General Development Settings assigns **Shift+Alt+F7** to `Window.PreviousDocumentWindowNav` and **Alt+F7** to `Window.NextDocumentWindowNav`.

NOTE

If the settings combination you are using does not already have a shortcut key combination assigned to this command, you can assign your own custom command using the **Keyboard** page of the **Options** dialog box. For more information, see [Identifying and Customizing Keyboard Shortcuts](#).

To switch to specific files in the editor

- Press **Ctrl+Tab** to display the **IDE Navigator**. Hold down the CTRL key and press TAB repeatedly until you select the file you intend to switch to.

TIP

To reverse the order in which you go through the **Active Files** list, hold down the **Ctrl+Shift** keys and press **Tab**.

- or -

- In the upper right corner of the editor, choose the **Active Files** button, and then select a file from the list to switch to.

- or -

- On the menu bar, choose **Window**, **Windows**.
- In the list, select the file you want to view and then choose **Activate**.

Navigating Among Tool Windows in the IDE

The **IDE Navigator** also lets you cycle through the tool windows you have open in the IDE. You can use either of two commands to access the **IDE Navigator** to cycle through tool windows, depending on the order in which you want to cycle through. `Window.PreviousToolWindowNav` allows you to move to the file most recently accessed and `Window.NextToolWindowNav` allows you to move in the reverse order. General Development Settings assigns **Shift+Alt+F7** to `Window.PreviousDocumentWindowNav` and **Alt+F7** to `Window.NextDocumentWindowNav`.

NOTE

If the settings combination you are using does not already have a shortcut key combination assigned to this command, you can assign your own custom command using the **Keyboard** page of the **Options** dialog box. For more information, see [Identifying and Customizing Keyboard Shortcuts](#).

To switch to a specific tool window in the IDE

- Press **Alt+F7** to display the **IDE Navigator**. Hold down the **Alt** key and press **F7** repeatedly until you select the window you intend to switch to.

TIP

To reverse the order in which you go through the **Active Tool Windows** list, hold down the **SHIFT + ALT** keys and press **F7**.

See also

[Customizing window layouts](#)

[Default Keyboard Shortcuts](#)

Solutions and projects in Visual Studio

12/22/2017 • 2 min to read • [Edit Online](#)

Projects

When you create an app, website, plug-in, etc. in Visual Studio, you start with a *project*. In a logical sense, a project contains all the source code files, icons, images, data files, etc. that are compiled into an executable, library, or website. A project also contains compiler settings and other configuration files that might be needed by various services or components that your program communicates with.

NOTE

You don't have to use solutions or projects in Visual Studio to edit, build and debug code. You can simply open the folder that contains your source files in Visual Studio and start editing. See [Develop code in Visual Studio without projects or solutions](#) for more information.

A project is defined in an XML file with an extension such as .vbproj, .csproj, or .vcxproj. This file contains a virtual folder hierarchy, and paths to all the items in the project. It also contains the build settings.

TIP

To look at the contents of a project file in Visual Studio, first unload the project by selecting the project name in Solution Explorer and choosing **Unload Project** from the context or right-click menu. Then, open the context menu again and choose **Edit <projectname>**.

In Visual Studio, the project file is used by Solution Explorer to display the project contents and settings. When you compile your project, the MSBuild engine consumes the project file to create the executable. You can also customize projects to produce other kinds of output.

Solutions

A project is contained within a *solution*. A solution contains one or more related projects, along with build information, Visual Studio window settings, and any miscellaneous files that aren't associated with a particular project. A solution is described by a text file (extension .sln) with its own unique format; it is generally not intended to be edited by hand.

A solution has an associated .suo file that stores settings, preferences and configuration information for each user that has worked on the project.

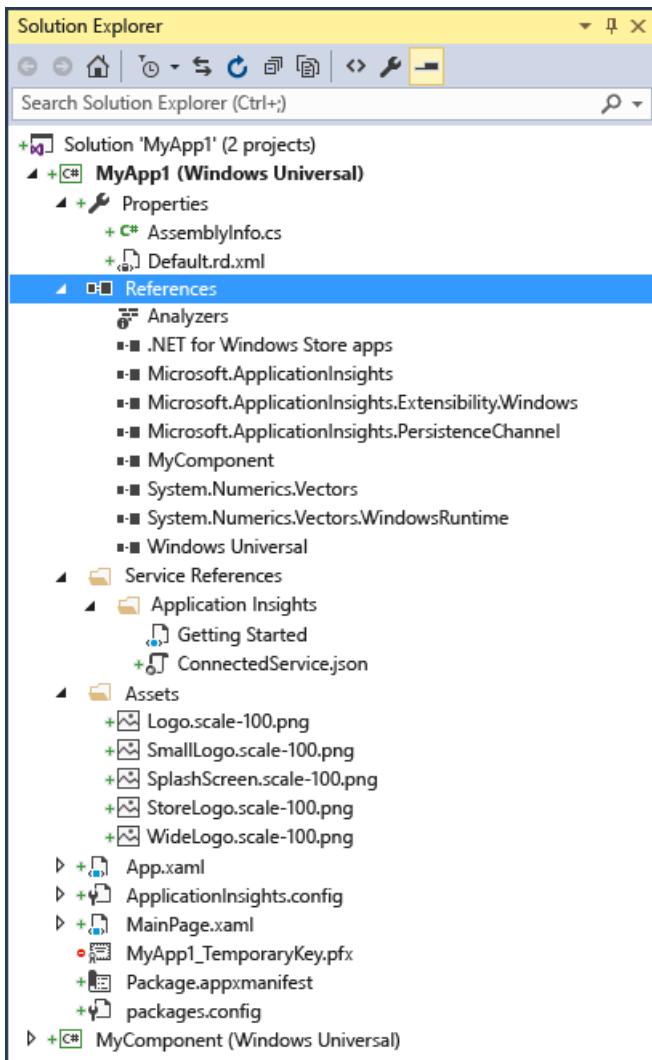
Creating new projects

The easiest way to create a new project is to start from a project template for a particular type of application or website. A project template consists of a basic set of pre-generated code files, config files, assets, and settings. These templates are what you see in the **New Project** or **New Web Site** dialog box when you choose **File, New, Project** or **File, New, Web Site**. For more information, see [Creating Solutions and Projects](#).

You can also create custom project and item templates. For more information, see [Creating Project and Item Templates](#).

Managing projects in Solution Explorer

After you create a new project, you can use **Solution Explorer** to view and manage the project and solution, and their associated items. The following illustration shows Solution Explorer with a C# solution that contains two projects.



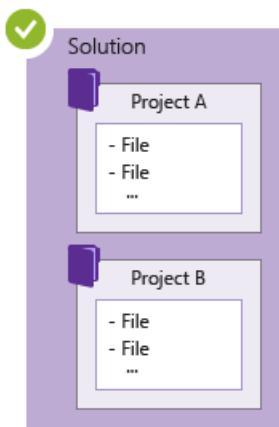
See also

[Visual Studio IDE](#)

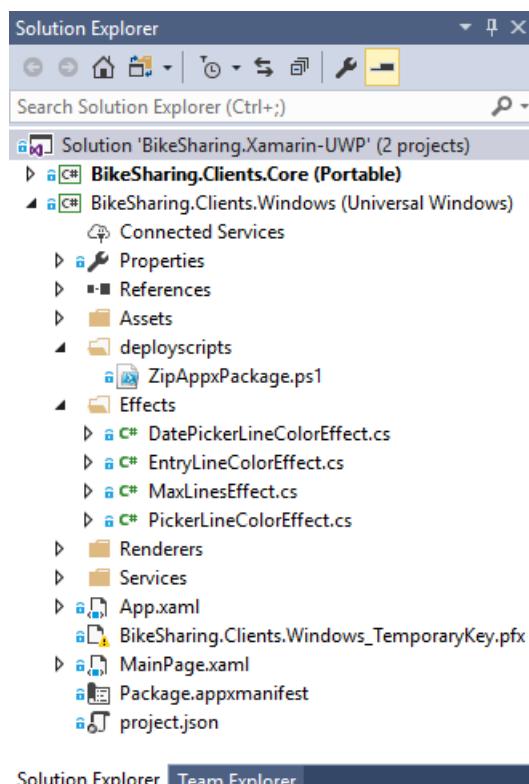
Create solutions and projects

3/5/2018 • 6 min to read • [Edit Online](#)

Projects are logical containers in Visual Studio that hold the items needed to build your app, such as source code files, bitmaps, icons, and component and service references. When you create a new project, Visual Studio creates a *solution* to contain the project. You can then add other new or existing projects to the solution if you want. Solutions can also contain files that aren't connected to any specific project.



You can view your solutions and projects in a tool window called **Solution Explorer**. The following screenshot shows an example solution in Solution Explorer (BikeSharing.Xamarin-UWP) that contains two projects: BikeSharing.Clients.Core and BikeSharing.Clients.Windows. Each project contains multiple files, folders, and references. The project name in bold is the *startup project*; that is, the project that starts when you run the app. You can specify which project is the startup project.



While you can construct a project yourself by adding the necessary files to it, Visual Studio offers a selection of project templates to give you a head start. Creating a new project from a template gives you a project with the essentials for that project type, and you can rename the files or add new or existing code and other resources to it.

as needed.

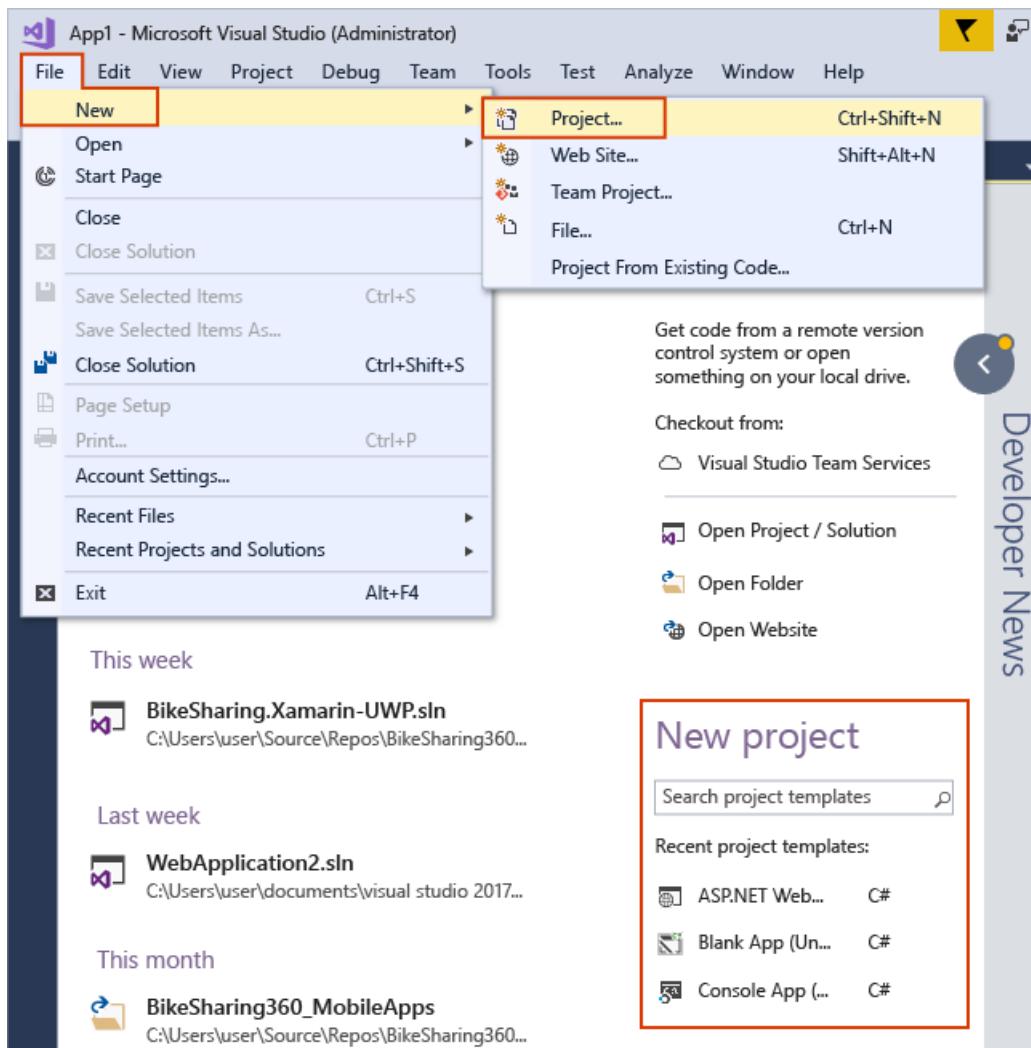
That being said, solutions and projects are not required to develop apps in Visual Studio. You can also just open code that you have cloned from Git or downloaded elsewhere. For more information, see [Develop code in Visual Studio without projects or solutions](#).

NOTE

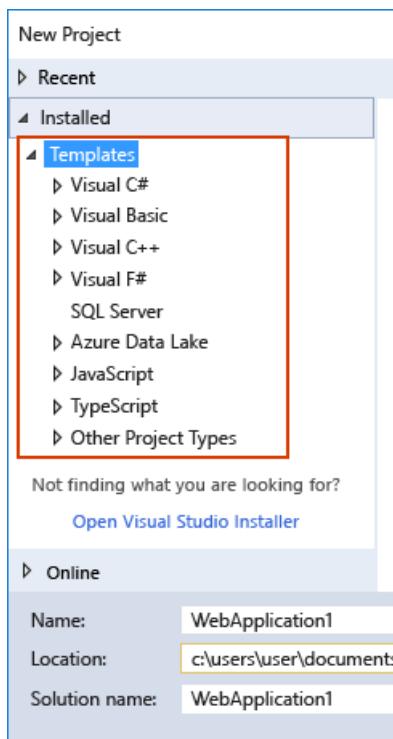
The descriptions in this topic are based on the Visual Studio Community edition. The dialog boxes and menu commands you see might differ from those described here, depending on your settings or Visual Studio edition. To change your settings, for example to **General** or **Visual C++** settings, choose **Tools**, **Import and Export Settings**, and then choose **Reset all settings**.

To create a project from a project template

1. There are multiple ways to create a new project in Visual Studio. On the Start Page, enter the name of a project template in the **Search project templates** box, or choose the **Create new project** link to open the **New Project** dialog box. You can also choose **File > New > Project...** on the menu bar, or choose the **New Project** button on the toolbar.



In the **New Project** dialog box, available project templates appear in a list under the **Templates** category. Templates are organized by programming language and project type, such as Visual C#, JavaScript, and Azure Data Lake.

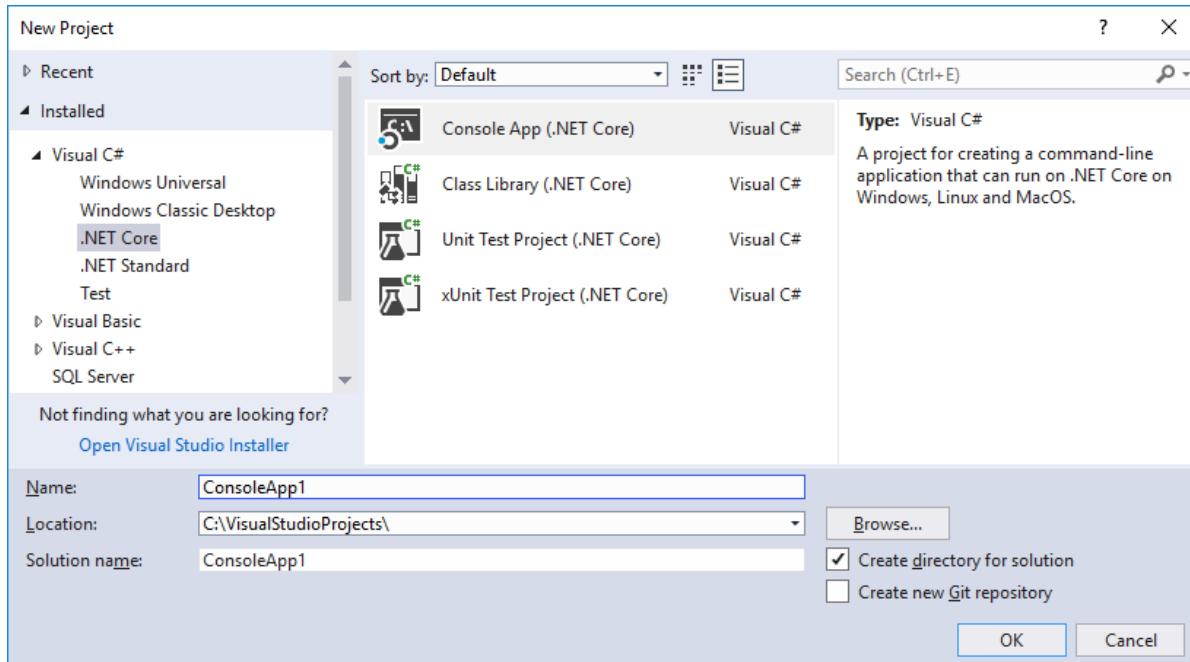


NOTE

The list of available languages and project templates that appears depends on the version of Visual Studio you are running and the workloads that are installed. To learn about how to install additional workloads, see [Modify Visual Studio 2017 by adding or removing workloads and components](#).

2. Show the list of templates for the programming language you want to use by choosing the triangle next to the language name, and then choose a project type.

The following example shows the project templates available for Visual C# .NET Core projects.



3. Enter a name for the new project in the **Name** box. You can choose to save the project in the default location on your system, or choose the **Browse** button to find another location.

You can also optionally choose to change the solution name, or add the new project to a Git repository by choosing **Add to Source Control**.

4. Choose the **OK** button to create the solution and project.
5. If you want to add an additional project to the solution, choose the solution node in Solution Explorer, and then on the menu bar, choose **Project > Add New Item**.

Create a project from existing code files

If you have a collection of code source files, you can easily add them to a project.

1. On the menu, choose **File > New > Project From Existing Code**.
2. In the **Create Project from Existing Code Files** wizard, choose the project type you want in the **What type of project would you like to create?** drop-down list box, and then choose the **Next** button.
3. In the wizard, browse to the location of the files and then enter a name for the new project in the **Name** box. When you are done, choose the **Finish** button.

NOTE

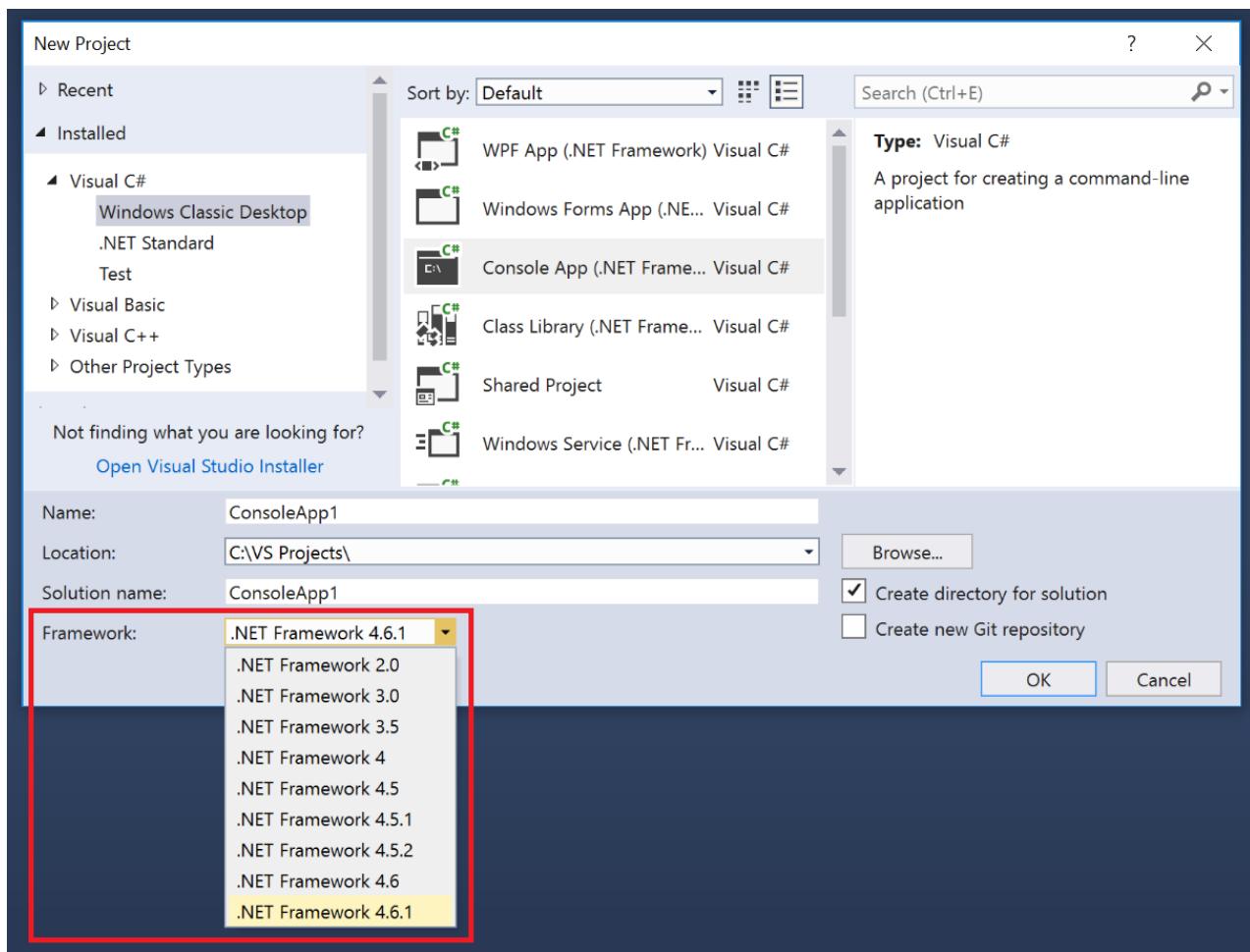
This option works best for a relatively simple collection of files. Currently, only Visual C++, Apache Cordova, Visual Basic, and C# project types are supported.

Add files to a solution

If you have a file that applies to multiple projects, such as a readme file for the solution, or other files that logically belong at the solution level rather than under a specific project, then you can add them to the solution itself. To add an item to a solution, on the context (right-click) menu of the solution node in **Solution Explorer**, choose **Add > New Item**, or **Add > Existing Item**.

Create a .NET project that targets a specific version of the .NET Framework

When you create a project, you can specify a specific version of the .NET Framework that you want the project to use. To specify a .NET framework version, choose the **Framework** drop-down menu in the **New Project** dialog box.



NOTE

You must have .NET Framework 3.5 installed on your system to access .NET Framework versions earlier than .NET Framework 4.

Create empty solutions

You can also create empty solutions that have no projects. This might be preferable in cases where you want to construct your solution and projects from scratch.

To create an empty solution

1. On the menu, choose **File > New > Project....**
2. In the left (**Templates**) pane, choose **Other Project Types > Visual Studio Solutions** in the expanded list.
3. In the middle pane, choose **Blank Solution**.
4. Enter **Name** and **Location** values for your solution, then choose **OK**.

After you create an empty solution, you can add new or existing projects or items to it by choosing **Add New Item** or **Add Existing Item** on the **Project** menu.

As mentioned earlier, you can also open code files without needing a project or solution. To learn about developing code in this way, see [Develop code in Visual Studio without projects or solutions](#).

Create a temporary project (C# and Visual Basic)

If you create a .NET-based project without specifying a disk location, it is a temporary project. Temporary projects enable you to experiment with .NET projects. At any time while you are working with a temporary project, you can

choose to save it or discard it.

To create a temporary project, first go to **Tools > Options > Projects and Solutions > General**, and uncheck the **Save new projects when created** checkbox. Then open the **New Project** dialog box as usual.

Delete a solution, project, or item

You can delete solutions and their contents permanently, but not by using the Visual Studio IDE. Deleting items within Visual Studio only removes them from the current solution or project. To permanently delete a solution or other component from your system, use File Explorer to delete the folder that contains the .sln and .suo solution files. However, before permanently deleting a solution, it's recommended that you back up any projects or files in case you need them again.

NOTE

The .suo file is a hidden file that is not displayed under the default File Explorer settings. To show hidden files, on the **View** menu in File Explorer, select the **Hidden Items** checkbox.

To permanently delete a solution

1. In **Solution Explorer**, on the context menu of the solution you want to delete, choose **Open folder in File Explorer**.
2. In File Explorer, navigate up one level.
3. Choose the folder containing the solution and then choose the **Delete** key.

See also

[Solutions and projects](#)

[Microsoft's open source repositories on GitHub](#)

[Visual Studio Samples](#)

[Developer code samples](#)

Port, Migrate, and Upgrade Visual Studio Projects

3/16/2018 • 12 min to read • [Edit Online](#)

Each new version of Visual Studio generally supports most previous types of projects, files, and other assets. You can work with them [as you always have](#), and provided that you don't depend on newer features, Visual Studio generally tries to preserve backwards compatibility with previous versions like Visual Studio 2015, Visual Studio 2013, and Visual Studio 2012. (See the [Release Notes](#) for which features are specific to which versions.)

Support for some project types also changes over time. A newer version of Visual Studio may no longer support certain projects, or requires updating a project such that it's no longer backwards compatible. For current status on migration issues, refer to the [Visual Studio Developer Community site](#).

IMPORTANT

This present article provides details only for project types in Visual Studio 2017 that involve migration. It does not include supported project types that have no migration issues; that list is found on [Platform Targeting and Compatibility](#). Note also that some project types are no longer supported in Visual Studio 2017 at all and therefore cannot be migrated.

IMPORTANT

Certain project types require installing the appropriate workloads through the Visual Studio installer. If you don't have the workload installed, Visual Studio reports an unknown or incompatible project type. In that case, check your installation options and try again. Again, see the [Platform Targeting and Compatibility](#) article for details on project support in Visual Studio 2017.

Project types

The following list describes support in Visual Studio 2017 for projects that were created in earlier versions.

If you don't see a project or file type listed here that should be, consult the [Visual Studio 2015 version of this article](#) and use the "Give documentation feedback" option at the bottom of this page to provide details of your project. (If you'd like a response, use the documentation feedback rather than the anonymous "Is this page helpful?" control.)

| TYPE OF PROJECT | SUPPORT |
|----------------------------|---|
| .NET Core projects (xproj) | Projects created with Visual Studio 2015 used preview tooling that included an xproj project file. When you open an xproj file with Visual Studio 2017, you are prompted to migrate the file to the csproj format (a backup of the xproj file is made). This csproj format for .NET Core projects is not supported in Visual Studio 2015 and earlier. The xproj format is not supported in Visual Studio 2017 other than for migration to csproj. For more information, see Migrating .NET Core projects to the csproj format . |

| TYPE OF PROJECT | SUPPORT |
|--|--|
| ASP.NET Web Application and ASP.NET Core Web Application with Application Insights enabled | <p>For each Visual Studio user, resource information is stored in the registry per user instance. This information is used when a user doesn't have a project opened and wants to search Azure Application Insights data. Visual Studio 2015 uses different registry location than Visual Studio 2017 and does not conflict.</p> <p>Once a user creates an ASP.NET Web Application or ASP.NET Core Web Application, the resource is stored in the .suo file. The user can open the project in Visual Studio 2015 or 2017 and the resource information is used for both as long as Visual Studio supports projects and solutions being used across both versions. Users need to authenticate once on each product. For example, if a project is created with Visual Studio 2015 and opened in Visual Studio 2017, the user needs to authenticate on Visual Studio 2017.</p> |
| C#/Visual Basic Webform or Windows Form | You can open the project in Visual Studio 2017 and Visual Studio 2015. |
| Database Unit Test Projects (csproj, .vbproj) | <p>Older Data Unit test projects are loaded in Visual Studio 2017 but use the GAC'd version of dependencies. To upgrade the unit test project to use the latest dependencies, right-click on the project in Solution Explorer and select Convert to SQL Server Unit Testing Project...</p> |
| F# | <p>Visual Studio 2017 can open projects created in Visual Studio 2013 and 2015. To enable Visual Studio 2017 features in these projects, however, open the project properties and change target fsharp.core to F# 4.1. Note also that the F# language support option in the Visual Studio installer is not selected by default with .NET workloads; you must include it by selecting that option for the workload, or selecting it from the Individual components tab under Development activities.</p> |
| InstallShield MSI setup | <p>Installer projects created in Visual Studio 2010 can be opened in later versions with the help of the Visual Studio Installer Projects extension. Also see the WiX Toolset Visual Studio 2017 Extension. InstallShield Limited Edition is no longer included with Visual Studio. Check with Flexera Software about availability for Visual Studio 2017.</p> |
| LightSwitch | <p>LightSwitch is no longer supported in Visual Studio 2017. Projects created with Visual Studio 2012 and earlier opened in Visual Studio 2013 or Visual Studio 2015 are upgraded and can be opened only in Visual Studio 2013 or Visual Studio 2015 thereafter.</p> |
| Microsoft Azure Tools for Visual Studio | <p>To open these types of projects, first install the Azure SDK for .NET, then open the project. If necessary, your project is updated.</p> |

| TYPE OF PROJECT | SUPPORT |
|---|--|
| Model-View-Controller framework (ASP.NET MVC) | <p>Support for MVC versions and Visual Studio:</p> <ul style="list-style-type: none"> Visual Studio 2010 SP1 supports MVC 2 and MVC 3; MVC 4 support is added through the ASP.NET 4 MVC 4 for Visual Studio 2010 SP1 download Visual Studio 2012 supports only MVC 3 and MVC 4 Visual Studio 2013 supports only MVC 4 and MVC 5 Visual Studio 2017 and Visual Studio 2015 supports MVC 4 (you can open existing projects but not create new ones) and MVC 5 <p>Upgrading MVC versions:</p> <ul style="list-style-type: none"> For information about how to automatically upgrade from MVC 2 to MVC 3, see ASP.NET MVC 3 Application Upgrader. For information about how to manually upgrade from MVC 2 to MVC 3, see Upgrading an ASP.NET MVC 2 Project to ASP.NET MVC 3 Tools Update. For information about how to manually upgrade from MVC3 to MVC 4, see Upgrading an ASP.NET MVC 3 Project to ASP.NET MVC 4. If your project targets .NET Framework 3.5 SP1, you must retarget it to use .NET Framework 4. For information about how to manually upgrade from MVC 4 to MVC 5, see How to Upgrade an ASP.NET MVC 4 and Web API Project to ASP.NET MVC 5 and Web API 2. |
| Modeling | <p>If you allow Visual Studio to update the project automatically, you can open it in Visual Studio 2015, Visual Studio 2013, or Visual Studio 2012.</p> <p>The format of the modeling project has not changed between Visual Studio 2015 and Visual Studio 2017 and the project can be opened and modified in either version. However, there are differences in behavior in Visual Studio 2017:</p> <ul style="list-style-type: none"> Modeling projects are now referred to as "Dependency Validation" projects in the menus and templates. UML diagrams are no longer supported in Visual Studio 2017. UML files are listed in the Solution Explorer as before but are opened as XML files. Use Visual Studio 2015 to view, create, or edit UML diagrams. In Visual Studio 2017, validation of architectural dependencies is no longer performed when the modeling project is built. Instead, validation is carried out as each code project is built. This change does not affect the modeling project, but it does require changes to the code projects being validated. Visual Studio 2017 can automatically make the necessary changes to the code projects (more information). |
| MSI Setup (vdproj) | See InstallShield Projects. |
| Office 2007 VSTO | Requires a one-way upgrade for Visual Studio 2017. |

| Type of Project | Support |
|---|---|
| Office 2010 VSTO | If the project targets the .NET Framework 4, you can open it in Visual Studio 2010 SP1 and later. All other projects require a one-way upgrade. |
| Service Fabric (sfproj) | Service Fabric Application Projects can be opened in either Visual Studio 2015 or Visual Studio 2017, unless the Service Fabric Application project references an ASP.NET Core service project. Service Fabric projects from Visual Studio 2015 that are opened in Visual Studio 2017 are one-way migrated from the xproj format to csproj. See ".NET Core projects (xproj)" earlier in this table. |
| SharePoint 2010 | <p>When a SharePoint solution project is opened with Visual Studio 2017, it's upgraded to either SharePoint 2013 or SharePoint 2016. The ".NET Desktop Development" workload must be installed in Visual Studio 2017 for the upgrade.</p> <p>For more information about how to upgrade SharePoint projects, see Upgrade to SharePoint 2013, Update Workflow in SharePoint Server 2013, and Create the SharePoint Server 2016 farm for a database attach upgrade.</p> |
| SharePoint 2016 | <p>SharePoint Add-In projects created in Office Developer Tools Preview 2 cannot be opened in Visual Studio 2017. To work around this limitation, update the <code>MinimumVisualStudioVersion</code> to 12.0 and <code>MinimumOfficeToolsVersion</code> to 12.2 in the csproj vbproj file.</p> |
| Silverlight | Silverlight projects not supported in Visual Studio 2017. To maintain Silverlight applications, continue to use Visual Studio 2015. |
| SQL Server Reporting Services and SQL Server Analysis Services (SSRS, SSDT, SSAS, MSAS) | Support for these project types is provided through two extensions in the Visual Studio Gallery: Microsoft Analysis Services Modeling Projects and Microsoft Reporting Services Projects . SSDT support is also included with the Data Storage and Processing workload in Visual Studio 2017. |
| SQL Server Integration Services (SSIS) | Support for Visual Studio 2017 is available through the SQL Server Data Tools (SSDT). For more information, see the SQL Server Integration Services blog . |
| Visual C++ | You can use Visual Studio 2017 to work in projects that were created in earlier versions of Visual Studio back to Visual Studio 2010. When you first open the project, you have the option to upgrade to the latest compiler and toolset or to continue using the original ones. If you choose to keep using the original ones, Visual Studio 2017 does not modify the project file, and uses the toolset from the earlier Visual Studio installation to build your project. Keeping the original options means you can still open the project in the original version of Visual Studio if necessary. For more information, see Use native multi-targeting in Visual Studio to build old projects . |

| TYPE OF PROJECT | SUPPORT |
|---|--|
| Visual Studio Extensibility/VSIX | Projects with MinimumVersion 14.0 or less are updated to declare MinimumVersion 15.0, which prevents the project from being opened in earlier versions of Visual Studio. To allow a project to open in earlier versions, set MinimumVersion to <code>\$(VisualStudioVersion)</code> . See also How to: Migrate Extensibility Projects to Visual Studio 2017 . |
| Visual Studio Lab Management | You can use Microsoft Test Manager or Visual Studio 2010 SP1 and later to open environments created in any of these versions. However, for Visual Studio 2010 SP1 the version of Microsoft Test Manager must match the version of Team Foundation Server before you can create environments. |
| Visual Studio Tools for Apache Cordova | Projects can be opened in Visual Studio 2017, but are not backwards compatible. Upon opening a project from Visual Studio 2015, you're prompted to allow modifications to your project. This modification upgrades the project to use toolsets instead of a <code>taco.json</code> file to manage the versioning of the Cordova library, its platforms, its plugins, and its node/npm dependencies. See the migration guide for more information. |
| Web Deployment (wdproj) | Support for Web Deployment projects was removed in Visual Studio 2012 with the addition of publish profile support. Because there is no equivalent in Visual Studio 2017, there is no automatic migration path for such projects. Instead, open the wdproj file in a text editor and copy-paste any customizations into the pubxml (publish profile) file, as described on StackOverflow . Also see Plans regarding website and web deployment projects (MSDN blogs) . |
| Windows Communication Foundation, Windows Workflow Foundation | You can open this project in Visual Studio 2017, Visual Studio 2015, Visual Studio 2013, and Visual Studio 2012 |
| Windows Presentation Foundation | You can open this project in Visual Studio 2013, Visual Studio 2012, and Visual Studio 2010 SP1. |
| Windows Store/Phone apps | Projects for Windows Store 8.1 and 8.0, and Windows Phone 8.1 and 8.0 are not supported in Visual Studio 2017. To maintain these apps, continue to use Visual Studio 2015. To maintain Windows Phone 7.x projects, use Visual Studio 2012. |

How Visual Studio decides when to migrate a project

Each new version of Visual Studio generally seeks to maintain compatibility with previous versions, such that the same project can be opened, modified, and built across different versions. However, there are inevitable changes over time such that some project types may no longer be supported. (See [Platform Targeting and Compatibility](#) for which project types are supported in Visual Studio 2017.) In these cases, a newer version of Visual Studio won't load the project and doesn't offer a migration path; you need to maintain that project in a previous version of Visual Studio that does support it.

In other cases, the newer version of Visual Studio can open a project, but must update or migrate the project in such a way that might render it incompatible with previous versions. Visual Studio uses a number of criteria to determine whether such migration is necessary:

- Compatibility with the target versions of platforms, back to Visual Studio 2013 RTM.

- Compatibility of design-time assets with previous versions of Visual Studio. (Namely different channels of Visual Studio 2017; Visual Studio 2015 RTM & Update 3; Visual Studio 2013 RTM & Update 5; Visual Studio 2012 Update 4; Visual Studio 2010 SP 1.) Visual Studio 2017 aims to fail gracefully with deprecated design-time assets without corrupting them, such that previous versions can still open the project.
- Whether new design time assets would break compatibility with previous versions down to Visual Studio 2013 RTM & Update 5.

The engineering owner of the project type in question looks at these criteria and makes the call where support, compatibility, and migration are concerned. Again, Visual Studio tries to maintain transparent compatibility between Visual Studio versions if possible, meaning that one can create and modify projects in one version of Visual Studio and it just works in other versions.

If such compatibility is not possible, however, as with some of the project types described in this article, then Visual Studio opens the upgrade wizard to make the necessary one-way changes.

Such one-way changes may involve changing the `ToolsVersion` property in the project file, which denotes exactly which version of MSBuild can turn the project's source code into runnable and deployable artifacts that you ultimately want. That is, what renders a project incompatible with previous versions of Visual Studio is not the *Visual Studio* version, but the *MSBuild* version, as determined by `ToolsVersion`. So long as your version of Visual Studio contains the MSBuild toolchain that matches the `ToolsVersion` in a project, then Visual Studio can invoke that toolchain to build the project.

To maintain maximum compatibility with projects created in older versions, Visual Studio 2017 includes the necessary MSBuild toolchains to support `ToolsVersion` 15, 14, 12, and 4. Projects that use any of these `ToolsVersion` values should result in a successful build. (Subject, again, to whether Visual Studio 2017 supports the project type at all, as described on [Platform Targeting and Compatibility](#).)

In this context, the question naturally arises whether you should try to manually update or migrate a project to a newer `ToolsVersion` value. Making such a change is unnecessary, and would likely generate many errors and warnings that you'd need to fix to get the project to build again. Furthermore, if Visual Studio drops support for a specific `ToolsVersion` in the future, then opening the project will trigger the project migration process specifically because the `ToolsVersion` value must be changed. In such a case, the subsystem for that specific project type knows exactly what needs to be changed, and can make those changes automatically as described earlier in this article.

Refer to the following articles for further discussion:

- [ToolsVersion guidance](#)
- [Framework targeting guidance](#)

4 min to read •

Project and solution file types

3/27/2018 • 1 min to read • [Edit Online](#)

Visual Studio supports many file types. In a particular installation, the installed components determine which file types are supported. This topic lists solution and project file types that are supported in some typical installations.

Solution files (.sln and .suo)

Visual Studio uses two file types (.sln and .suo) to store settings for solutions. These files, known collectively as solution files, provide Solution Explorer with the information it needs to display a graphical interface for managing your files.

| Extension | Name | Description |
|-----------|------------------------|---|
| .sln | Visual Studio Solution | Organizes projects, project items and solution items into the solution. |
| .suo | Solution User Options | Keeps track of user-level customizations you have made to Visual Studio, such as breakpoints. |

Project files

Projects can contain many different file types. For example, C# code files have a **.cs** extension, and C++ files have a **.cpp** extension. Resources are stored in **.resx** files, and XAML in **.xaml** files. **App.config** files contain application information that should not be included in the application code—for example connection strings.

For more information about file types in C++ projects, see [File Types Created for Visual C++ Projects](#).

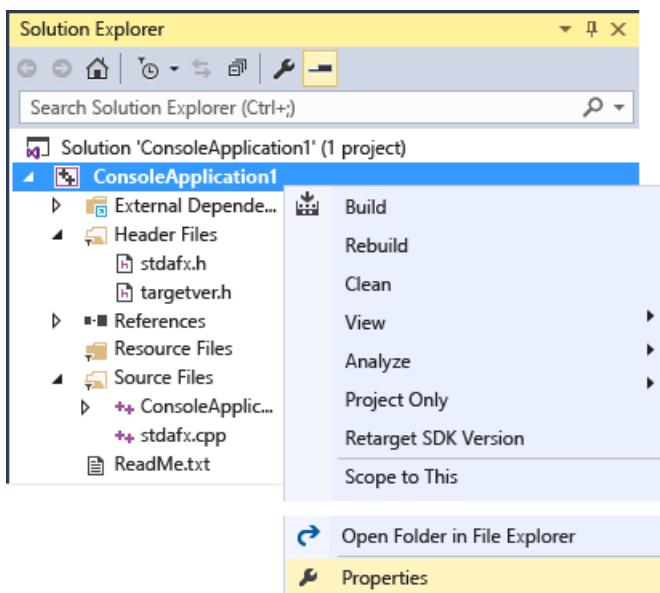
See also

[Solutions and Projects](#)

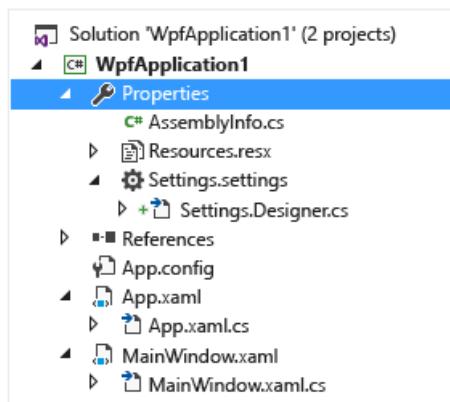
Managing project and solution properties

12/22/2017 • 1 min to read • [Edit Online](#)

Projects have properties that govern many aspects of compilation, debugging, testing and deploying. Some properties are common among all project types, and some are unique to specific languages or platforms. You access project properties by right-clicking the project node in Solution Explorer and choosing **Properties**, or by typing "properties" into the **Quick Launch** search box on the menu bar.



.NET projects might also have a properties node in the project tree itself.

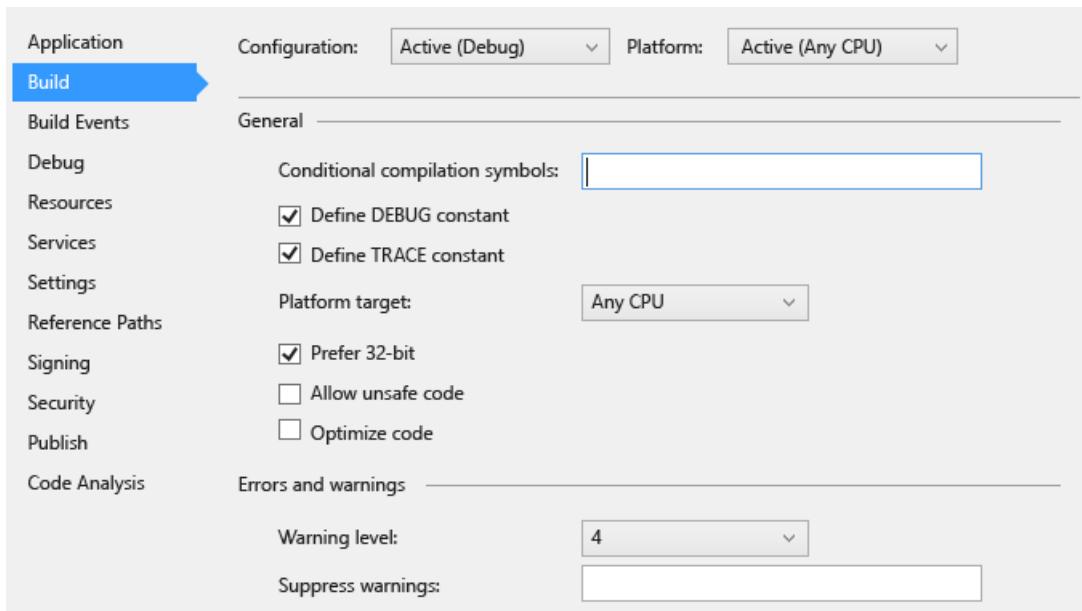


Project properties

Project properties are organized into groups, and each group has its own property page. The pages might be different for different languages and project types.

C#, Visual Basic and F# projects

In C#, Visual Basic and F# projects, properties are exposed in the **Project Designer**. The following illustration shows the Build property page for a WPF project in C#:



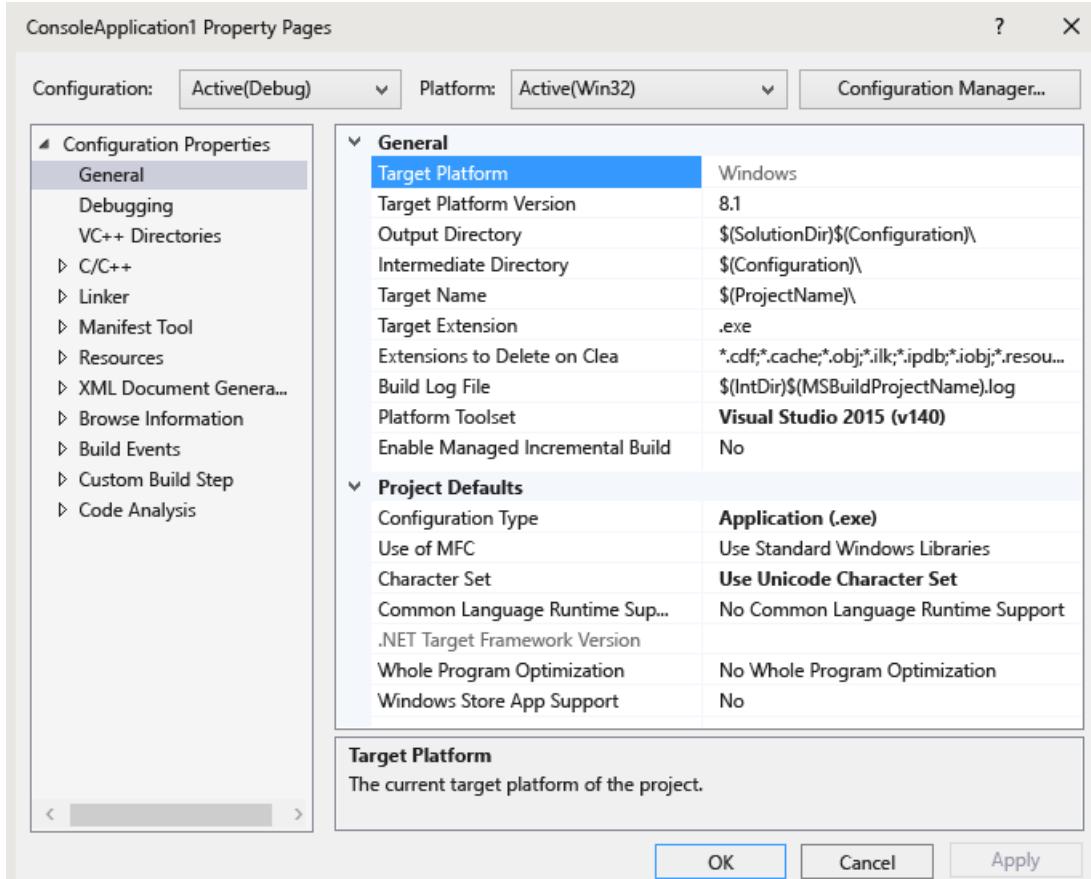
For information about each of the property pages in Project Designer, see [Project Properties Reference](#).

TIP

Solutions have a few properties, and so do project items; these properties are accessed in the [Properties Window](#), not [Project Designer](#).

C++ and JavaScript projects

C++ and JavaScript projects have a different user interface for managing project properties. This illustration shows a C++ project property page (JavaScript pages are similar):



For information about C++ project properties, see [Working with Project Properties \(C++\)](#). For more information about JavaScript properties, see [Property Pages, JavaScript](#).

Solution properties

To access properties on the solution, right click the solution node in **Solution Explorer** and choose **Properties**. In the dialog, you can set project configurations for Debug or Release builds, choose which projects should be the startup project when F5 is pressed, and set code analysis options.

See also

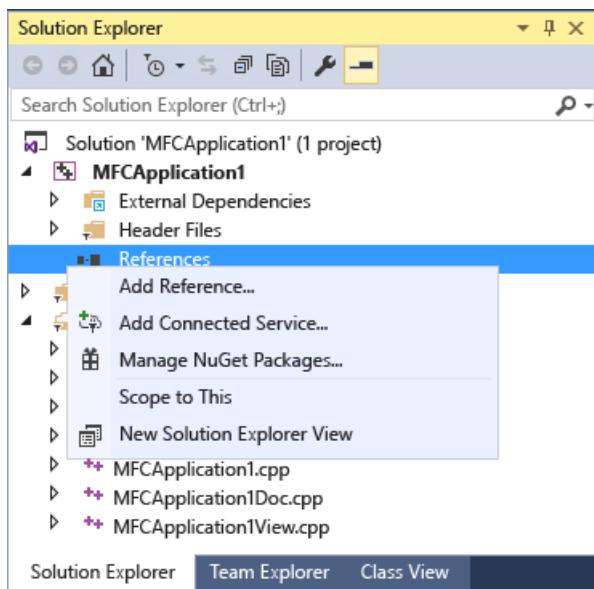
[Solutions and Projects in Visual Studio](#)

Managing references in a project

3/12/2018 • 6 min to read • [Edit Online](#)

Before you write code against an external component or connected service, your project must first contain a reference to it. A reference is essentially an entry in a project file that contains the information that Visual Studio needs to locate the component or the service.

To add a reference, right click on the References node in Solution Explorer and choose **Add Reference**. For more information, see [How to: Add or Remove References By Using the Reference Manager](#).



You can make a reference to the following types of components and services:

- .NET Framework class libraries or assemblies
- UWP apps
- COM components
- Other assemblies or class libraries of projects in the same solution
- XML Web services

UWP app references

Project references

Universal Windows Platform (UWP) projects can create references to other UWP projects in the solution, or to Windows 8.1 projects or binaries, provided that these projects do not use APIs that have been deprecated in Windows 10. For more information, see [Move from Windows Runtime 8 to UWP](#).

If you choose to retarget Windows 8.1 projects to Windows 10, see [Port, Migrate, and Upgrade Visual Studio Projects](#).

Extension SDK references

Visual Basic, C#, C++ and JavaScript Universal Windows Platform (UWP) apps can reference Extension SDKs that target Windows 8.1, as long as these Extension SDKs do not use APIs that have been deprecated in Windows 10. Please check the Extension SDK vendor site to find out whether it can be referenced by UWP apps.

If you determine that the Extension SDK being referenced by your app is not supported, then you need to perform the following steps:

1. Look at the name of the project that is causing the error. The platform your project is targeting is noted in parentheses next to the project name. For example, **MyProjectName (Windows 8.1)** means that your project **MyProjectName** is targeting platform version Windows 8.1.
2. Go to the site of the vendor who owns the unsupported Extension SDK, and install the version of the Extension SDK with dependencies that are compatible with the version of the platform your project is targeting.

NOTE

One way to find out whether an Extension SDK has dependencies on other Extension SDKs is by looking in **Reference Manager**. Restart Visual Studio, create a new C# UWP app project, and then right-click on the project and choose **Add Reference**. Go to the **Windows** tab, then the **Extensions** sub-tab, and select the Extension SDK. Look at the right pane in the **Reference Manager**. If it has dependencies, they will be listed there.

IMPORTANT

If your project is targeting Windows 10, and the Extension SDK installed in the previous step has a dependency on the Microsoft Visual C++ Runtime Package, the version of Microsoft Visual C++ Runtime Package that is compatible with Windows 10 is v14.0, and is installed with Visual Studio.

3. If the Extension SDK you installed in the previous step has dependencies on other Extension SDKs, go to the sites of the vendors who own the dependencies, and install the versions of these dependencies that are compatible with the version of the platform your project is targeting.
4. Restart Visual Studio and open your app.
5. Right-click on the **References** node in the project that caused the error and choose **Add Reference**.
6. Click the **Windows** tab and then the **Extensions** sub-tab, then uncheck the checkboxes for the old Extension SDKs, and check the checkboxes for the new Extension SDKs. Click **OK**.

Adding a reference at design time

When you make a reference to an assembly in your project, Visual Studio searches for the assembly in the following locations:

- The current project directory. (You can find these assemblies by using the **Browse** tab.)
- Other project directories in the same solution. (You can find these assemblies on the **Projects** tab.)

NOTE

All projects contain an implied reference to mscorelib. Visual Basic projects contain an implied reference to `Microsoft.VisualBasic`. All projects contain an implied reference to `System.Core`, even if `System.Core` is removed from the list of references.

References to shared components at run time

At run time, components must be either in the output path of the project or in the Global Assembly Cache (GAC). If the project contains a reference to an object that is not in one of these locations, you must copy the reference to the output path of the project when you build the project. The **CopyLocal** property indicates whether this copy has

to be made. If the value is **True**, the reference is copied to the project directory when you build the project. If the value is **False**, the reference is not copied.

If you deploy an application that contains a reference to a custom component that is registered in the GAC, the component will not be deployed with the application, regardless of the [CopyLocal](#) setting. In earlier versions of Visual Studio, you could set the [CopyLocal](#) property on a reference to ensure that the assembly was deployed. Now, you must manually add the assembly to the \Bin folder. This puts all custom code under scrutiny, reducing the risk of publishing custom code with which you are not familiar.

By default, the [CopyLocal](#) property is set to **False** if the assembly or component is in the global assembly cache or is a framework component. Otherwise, the value is set to **True**. Project-to-project references are always set to **True**.

Referencing a project or assembly that targets a different version of the .NET Framework

You can create applications that reference projects or assemblies that target a different version of the .NET Framework. For example, you could create an application that targets the .NET Framework 4 Client Profile, that references an assembly that targets .NET Framework 2.0. If you create a project that targets an earlier version of the .NET Framework, you cannot set a reference in that project to a project or assembly that targets a newer version.

For more information, see [Multi-targeting overview](#).

Project-to project references

Project-to-project references are references to projects that contain assemblies; you create them by using the **Project** tab. Visual Studio can find an assembly when given a path to the project.

When you have a project that produces an assembly, you should reference the project and not use a file reference (see below). The advantage of a project-to-project reference is that it creates a dependency between the projects in the build system. The dependent project will be built if it has changed since the last time the referencing project was built. A file reference does not create a build dependency, so it is possible to build the referencing project without building the dependent project, and the reference can become obsolete. (That is, the project can reference a previously built version of the project.) This can result in several versions of a single DLL being required in the bin directory, which is not possible. When this conflict occurs, you will see a message such as "Warning: the dependency 'file' in project 'project' cannot be copied to the run directory because it would overwrite the reference 'file.'". For more information, see [Troubleshooting Broken References](#) and [How to: Create and Remove Project Dependencies](#).

NOTE

A file reference instead of a project-to-project reference is created if the target version of the .NET Framework of one project is version 4.5, and the target version of the other project is version 2, 3, 3.5, or 4.0.

File references

File references are direct references to assemblies outside the context of a Visual Studio project. You create them by using the **Browse** tab of the **Reference Manager**. Use a file reference when you just have an assembly or component, and not the project that creates it as output.

See also

[Troubleshooting Broken References](#) [How to: Add or Remove References By Using the Reference Manager](#)

How to: add or remove references by using the Reference Manager

3/12/2018 • 12 min to read • [Edit Online](#)

You can use the **Reference Manager** dialog box to add and manage references to components that you, Microsoft, or another company developed. If you're developing a Universal Windows app, your project automatically references all of the correct Windows SDK DLLs. If you are developing a .NET application, your project automatically references mscorelib.dll. Some .NET APIs are exposed in components that you have to add manually. References to COM components or custom components have to be added manually.

Reference Manager dialog box

The **Reference Manager** dialog box shows different categories on the left side, depending on the project type:

- Assemblies, with the Framework and Extensions subgroups.
- COM, lists all COM components that are available for referencing.
- Solution, with the Projects subgroup.
- Windows, with the Core and Extensions subgroups. You can explore the references in the Windows SDK or extension SDKs by using the **Object Browser**.
- Browse, with the Recent subgroup.

Adding and removing a reference

To add a reference

1. In **Solution Explorer**, right-click on the References node and choose **Add Reference**.
2. Specify the references to add, and then choose the **OK** button.

Reference Manager opens and lists the available references by group.

Assemblies tab

The **Assemblies** tab lists all .NET Framework assemblies that are available for referencing. The **Assemblies** tab doesn't list any assemblies from the global assembly cache (GAC) because assemblies in the GAC are part of the run-time environment. If you deploy or copy an application that contains a reference to an assembly that's registered in the GAC, the assembly won't be deployed or copied with the application, regardless of the Copy Local setting. For more information, see [Managing references in a project](#).

When you manually add a reference to any of the EnvDTE namespaces (EnvDTE, EnvDTE80, EnvDTE90, EnvDTE90a, or EnvDTE100), set the **Embed Interop Types** property of the reference to **False** in the Properties window. Setting this property to **True** can cause build issues because of certain EnvDTE properties that can't be embedded.

All desktop projects contain an implicit reference to mscorelib. Visual Basic projects contain an implicit reference to Microsoft.VisualBasic. All projects contain an implicit reference to System.Core, even if it's removed from the list of references.

If a project type doesn't support Assemblies, the tab won't appear in the **Reference Manager** dialog box.

The Assemblies tab consists of two sub-tabs:

1. **Framework** lists all assemblies that constitute the targeted Framework.

Projects for Windows 8.x Store apps contain references to all of the assemblies in the targeted .NET for Windows 8.x apps by default on project creation. In managed projects, a read-only node under the References folder in **Solution Explorer** indicates the reference to the entire Framework. Accordingly, the Framework tab won't enumerate any of the assemblies from the Framework and instead display the following message: "All of the Framework assemblies are already referenced. Please use the Object Browser to explore the references in the Framework." For desktop projects, the Framework tab enumerates assemblies from the targeted Framework, and the user must add the references that the application requires.

2. **Extensions** lists all assemblies that external vendors of components and controls have developed to extend the targeted Framework. Depending on the purpose of the user application, it might need these assemblies.

Extensions is populated by enumerating the assemblies that are registered in the following locations:

32-bit machine:

- HKEY_CURRENT_USER\SOFTWARE\Microsoft[Target Framework Identifier]\v[Target Framework Version]\AssemblyFoldersEx[UserComponentName]@default=[Disk location of assemblies]
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft[Target Framework Identifier]\v[Target Framework Version]\AssemblyFoldersEx[UserComponentName]@default=[Disk location of assemblies]

64-bit machine:

- HKEY_CURRENT_USER\SOFTWARE\Wow6432Node\Microsoft[Target Framework Identifier]\v[Target Framework Version]\AssemblyFoldersEx[UserComponentName]@default=[Disk location of assemblies]
- HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft[Target Framework Identifier]\v[Target Framework Version]\AssemblyFoldersEx[UserComponentName]@default=[Disk location of assemblies]
And older versions of the [Target Framework Identifier]

For example, if a project targets the .NET Framework 4 on a 32-bit machine, Extensions will enumerate assemblies that are registered under \Microsoft\.NETFramework\v4.0\AssemblyFoldersEx\, \Microsoft\.NETFramework\v3.5\AssemblyFoldersEx\, \Microsoft\.NETFramework\v3.0\AssemblyFoldersEx\, and \Microsoft\.NETFramework\v2.0\AssemblyFoldersEx\.

Some components in the list may not be shown, depending on the .NET Framework version of your project. This can occur under the following conditions:

- A component that uses a recent version of the .NET Framework is incompatible with a project that targets an earlier version of the .NET Framework.

For information about how to change the target .NET Framework version for a project, see [How to: Target a Version of the .NET Framework](#).

- A component that uses .NET Framework 4 is incompatible with a project that targets the .NET Framework 4.5.

When you create a new application, some projects target the .NET Framework 4.5 by default.

- You should avoid adding file references to outputs of another project in the same solution, because doing this may cause compilation errors. Instead, use the **Projects** tab of the **Add Reference** dialog box to create project-to-project references. This makes team development easier by enabling better management of the class libraries you create in your projects. For more information, see [Troubleshooting Broken References](#).

NOTE

In Visual Studio 2015 or later, a file reference instead of a project reference is created if the target version of the .NET Framework of one project is version 4.5 or later, and the target version of the other project is version 2, 3, 3.5, or 4.0.

To display an assembly in the Add Reference dialog box

- Move or copy the assembly to one of the following locations:
 - The current project directory. (You can find these assemblies by using the **Browse** tab.)
 - Other project directories in the same solution. (You can find these assemblies by using the **Projects** tab.)
- or -
- Set a registry key that specifies the location of assemblies to display:

For a 32-bit operating system, add one of the following registry keys.

- [HKEY_CURRENT_USER\SOFTWARE\Microsoft\.NETFramework\VersionMinimum\AssemblyFoldersEx\MyAssemblies]@="*AssemblyLocation*"
- [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\VersionMinimum\AssemblyFoldersEx\MyAssemblies]@="*AssemblyLocation*"

For a 64-bit operating system, add one of the following registry keys in a 32-bit registry hive.

- [HKEY_CURRENT_USER\SOFTWARE\Wow6432Node\Microsoft\.NETFramework\VersionMinimum\AssemblyFoldersEx\MyAssemblies]@="*AssemblyLocation*"
- [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\.NETFramework\VersionMinimum\AssemblyFoldersEx\MyAssemblies]@="*AssemblyLocation*"

VersionMinimum is the lowest .NET Framework version that applies. If *VersionMinimum* is v3.0, folders specified in AssemblyFoldersEx apply to projects that target .NET Framework 3.0 and later.

AssemblyLocation is the directory of the assemblies that you want to appear in the **Add Reference** dialog box, for example, C:\MyAssemblies\.

Creating the registry key under the HKEY_LOCAL_MACHINE node allows all users to see the assemblies in the specified location in the **Add Reference** dialog box. Creating the registry key under the HKEY_CURRENT_USER node affects only the setting for the current user.

Open the **Add Reference** dialog box again. The assemblies should appear on the **.NET** tab. If they do not, make sure that the assemblies are located in the specified *AssemblyLocation* directory, restart Visual Studio, and try again.

COM tab

The COM tab lists all COM components that are available for referencing. If you want to add a reference to a registered COM DLL that contains an internal manifest, unregister the DLL first. Otherwise, Visual Studio adds the assembly reference as an ActiveX Control instead of as a native DLL.

If a project type doesn't support COM, the tab won't appear in the **Reference Manager** dialog box.

Solution tab

The Solution tab lists all compatible projects within the current solution, in the Projects sub-tab.

A project can reference another project that targets a different version of the .NET Framework. For example, you could create a project that targets the .NET Framework 4 but that references an assembly that's been built for the .NET Framework 2. However, the .NET Framework 2 project can't reference a .NET Framework 4 project. For more information, see [Multi-targeting overview](#).

A project that targets the .NET Framework 4 is incompatible with a project that targets the .NET Framework 4 Client Profile.

A file reference is created instead of a project reference if one project targets the .NET Framework 4 and another project targets an earlier version.

A project that targets .NET for Windows 8.x apps can't add a project reference to a project that targets the .NET Framework, and vice versa.

Windows tab

The Windows tab lists all SDKs that are specific to platforms on which Windows operating systems run.

You can generate a WinMD file in Visual Studio in two ways:

- **Windows 8.x Store app managed projects:** Windows 8.x Store app projects can output WinMD binaries by setting Project Properties | Output Type = WinMD File. The WinMD filename must be the superset namespace of all the namespaces that exist within it. For example, if a project consists of namespaces A.B and A.B.C, the possible names for its outputted WinMD are A.winmd and A.B.winmd. If a user enters a Project Properties | Assembly Name or Project Properties | Namespace value that's disjoint from the set of namespaces in the project or there is no superset namespace within a project, a build warning is generated: 'A.winmd' isn't a valid .winmd file name for this assembly. All types within a Windows Metadata file must exist in a sub namespace of the file name. Types that don't exist in a sub namespace of the file name won't be able to be located at runtime. In this assembly, the smallest common namespace is 'CSWSClassLibrary1'. A desktop Visual Basic or C# project can only consume WinMDs that are generated by using the Windows 8 SDKs, which are known as first-party WinMDs, and can't generate WinMDs.
- **Windows 8.x Store app native projects:** A native WinMD file consists of only metadata. Its implementation exists in a separate DLL file. One can produce native binaries by choosing the Windows Runtime Component project template in the **New Project** dialog box or by starting from a blank project and modifying the project properties to generate a WinMD file. If the project consists of disjoint namespaces, a build error will tell the user to combine their namespaces or run the MSMerge tool.

The Windows tab consists of two subgroups.

Core subgroup

The Core subgroup lists all of the WinMDs (for Windows Runtime elements) in the SDK for the targeted version of Windows.

Windows 8.x Store app projects contain references to all of the WinMDs in the Windows 8 SDK by default on project creation. In managed projects, a read-only node under the References folder in **Solution Explorer** indicates the reference to the entire Windows 8 SDK. Accordingly, the Core subgroup in the Reference Manager won't enumerate any of the assemblies from the Windows 8 SDK and instead displays a message: "The Windows SDK is already referenced. Please use the Object Browser to explore the references in the Windows SDK."

In the desktop projects, the Core subgroup doesn't appear by default. You can add the Windows Runtime by opening the shortcut menu for the project node, choosing **Unload Project**, adding the following snippet, and re-opening the project (on the project node, choose **Reload Project**). When you invoke the **Reference Manager** dialog box, the Core subgroup appears.

```
<PropertyGroup>
  <TargetPlatformVersion>8.0</TargetPlatformVersion>
</PropertyGroup>
```

Make sure to select the **Windows** check box on this subgroup. You should then be able to use Windows Runtime elements. However, you'll also want to add System.Runtime, in which the Windows Runtime defines some standard classes and interfaces, such as `IEnumerable`, that are used throughout the Windows Runtime libraries. For information about how to add System.Runtime, see [Managed desktop apps and Windows Runtime](#).

Extensions subgroup

Extensions lists the user SDKs that extend the targeted Windows platform. This tab appears for Windows 8.x Store app projects only. Desktop projects won't show this tab because they can consume only first-party .winmd files.

An SDK is a collection of files that Visual Studio treats as a single component. In the Extensions tab, SDKs that apply to the project from which the **Reference Manager** dialog box was invoked are listed as single entries. When added to a project, all of the SDK content is consumed by Visual Studio such that the user doesn't need to take any further actions to leverage the SDK contents in IntelliSense, toolbox, designers, Object Browser, build, deployment, debugging, and packaging. For information about how to display your SDK in the Extensions tab, see [Creating a Software Development Kit](#).

NOTE

If a project references an SDK that depends on another SDK, Visual Studio won't consume the second SDK unless the user manually adds a reference to the second SDK. When a user chooses an SDK on the **Extensions** tab, the **Reference Manager** dialog box helps the user identify SDK dependencies by listing not only the name and version of the SDK but also the name of any SDK dependencies in the details pane. If a user doesn't notice the dependencies and only adds that SDK, MSBuild will prompt the user to add the dependencies.

If a project type doesn't support **Extensions**, the tab doesn't appear in the **Reference Manager** dialog box.

Browse button

You can use the **Browse** button to browse for a component in the file system.

A project can reference a component that targets a different version of the .NET Framework. For example, you could create an application that targets the .NET Framework 4.7, which references a component that targets the .NET Framework 4. For more information, see [Multi-targeting overview](#).

You should avoid adding file references to outputs of another project in the same solution, because this tactic may cause compilation errors. Instead, use the **Solution** tab of the **Reference Manager** dialog box to create project-to-project references. This makes team development easier by enabling better management of the class libraries that you create in your projects. For more information, see [Troubleshooting Broken References](#).

You can't browse to an SDK and add it to your project. You can only browse to a file (for example, an assembly or .winmd) and add it to your project.

When doing a file reference to a WinMD, the expected layout is that the `FileName.winmd`, `FileName.dll`, and `FileName.pri` files are all placed alongside each other. If you reference a WinMD in the following scenarios, an incomplete set of files will be copied into the project output directory and, consequently, build and runtime failures will occur.

- **Native component:** a native project will create one WinMD for each disjoint set of namespaces and one DLL that consists of the implementation. The WinMDs will have disparate names. When referencing this native component file, MSBuild won't recognize that the dissimilarly named WinMDs make one component. Consequently, only the identically named `FileName.dll` and `FileName.winmd` will be copied, and runtime

errors will occur. To work around this issue, create an Extension SDK. For more information, see [Creating a Software Development Kit](#).

- **Consuming controls:** at a minimum, a XAML control consists of a *FileName.winmd*, *FileName.dll*, *FileName.pri*, *XamlName.xaml*, and an *ImageName.jpg*. When the project is built, the resource files that are associated with the file reference won't get copied into the project's output directory, and only *FileName.winmd*, *FileName.dll* and *FileName.pri* will be copied. A build error is logged to inform the user that the resources *XamlName.xaml* and *ImageName.jpg* are missing. To succeed, the user will have to manually copy these resource files into the project output directory for build and debugging/runtime. To work around this issue, either create an Extension SDK by following the steps in [Creating a Software Development Kit](#) or edit the project file to add the following property:

```
<PropertyGroup>
  <GenerateLibraryOutput>True</GenerateLibraryOutput>
</PropertyGroup>
```

NOTE

If you add the property, the build might run slower.

Recent

Assemblies, COM, Windows, and Browse each support a Recent tab, which enumerates the list of components that were recently added to projects.

Search

The search bar in the **Reference Manager** dialog box operates over the tab that's in focus. For example, if a user types "System" in the search bar while the **Solution** tab is in focus, the search won't return any results unless the solution consists of a project name that contains "System".

See also

[Managing references in a project](#)

Adding References Using NuGet Versus an Extension SDK

2/20/2018 • 6 min to read • [Edit Online](#)

You can provide a package for consumption within Visual Studio projects by using either the NuGet extension to Visual Studio or a software development kit (SDK). By describing the similarities and differences between the two mechanisms, this topic can help you choose the best one for your task.

- NuGet is an open-source, package-management system that simplifies the process of incorporating libraries into a project solution. For more information, see the [NuGet documentation](#).
- An SDK is a collection of files that Visual Studio treats as a single reference item. The **Reference Manager** dialog box lists all SDKs that are relevant to the project that's open when you display that dialog box. When you add an SDK to a project, you can access all of the contents of that SDK through IntelliSense, the **Toolbox**, designers, the **Object Browser**, MSBuild, deployment, debugging, and packaging. For more information about SDKs, see [Creating a Software Development Kit](#).

Which Mechanism Should I Use?

The following table helps you compare the referencing features of an SDK with the referencing features of NuGet.

| FEATURE | SDK SUPPORT | SDK NOTES | NUGET SUPPORT | NUGET NOTES |
|---|-------------|---|---------------|-------------|
| The mechanism references one entity and then all the files and functionality are available. | Y | You add an SDK by using the Reference Manager dialog box, and all the files and functionality are available during the development workflow. | Y | |
| MSBuild automatically consumes assemblies and Windows metadata (.winmd) files. | Y | References in the SDK are automatically passed to the compiler. | Y | |
| MSBuild automatically consumes the .h or .lib files. | Y | The <i>SDKName.props</i> file tells Visual Studio how to set up the Visual C++ directory, and so forth, for automatic .h or .lib file consumption. | N | |

| FEATURE | SDK SUPPORT | SDK NOTES | NUGET SUPPORT | NUGET NOTES |
|---|-------------|---|---------------|---|
| MSBuild automatically consumes the .js or .css files. | Y | In Solution Explorer , you can expand the JavaScript SDK reference node to show individual .js or .css files and then generate <code><source include/></code> tags by dragging those files to their source files. The SDK supports F5 and automatic package setup. | Y | |
| MSBuild automatically adds the control in the Toolbox . | Y | The Toolbox can consume SDKs and show controls in the tabs that you specify. | N | |
| The mechanism supports Visual Studio Installer for extensions (VSIX). | Y | VSIX has a special manifest and logic to create SDK packages | Y | The VSIX can be embedded in another setup program. |
| The Object Browser enumerates references. | Y | The Object Browser automatically gets the list of references in SDKs and enumerates them. | N | |
| Files and links automatically get added to the Reference Manager dialog box (help links, and so forth auto populate) | Y | The Reference Manager dialog box automatically enumerates SDKs, along with help links and the list of SDK dependencies. | N | NuGet provides its own Manage NuGet Packages dialog box. |
| The mechanism supports multiple architectures. | Y | SDKs can ship multiple configurations. MSBuild consumes the appropriate files for each project configuration. | N | |
| The mechanism supports multiple configurations. | Y | SDKs can ship multiple configurations. Depending on project architecture, MSBuild consumes the appropriate files for each project architecture. | N | |

| FEATURE | SDK SUPPORT | SDK NOTES | NUGET SUPPORT | NUGET NOTES |
|--|-------------|---|---------------|---|
| The mechanism can specify "not to copy." | Y | Depending on whether files are dropped in the \redist or \designtime folder, you can control which files to copy into the consuming application's package. | N | You declare which files to copy in the package manifest. |
| Content appears in localized files. | Y | Localized XML documents in SDKs are automatically included for a better design-time experience. | N | |
| MSBuild supports consuming multiple versions of an SDK simultaneously. | Y | The SDK supports consuming multiple versions simultaneously. | N | This isn't referencing. You can't have more than one version of NuGet files in your project at a time. |
| The mechanism supports specifying applicable target frameworks, Visual Studio versions, and project types. | Y | The Reference Manager dialog box and the Toolbox show only the SDKs that apply to a project, so that users can more easily choose the appropriate SDKs. | Y (partial) | Pivot is the Target Framework. There is no filtering on user interface. At installation time, it might return an error. |
| The mechanism supports specifying registration info for native WinMDs. | Y | You can specify the correlation between the .winmd file and the .dll file in SDKManifest.xml. | N | |
| The mechanism supports specifying dependencies on other SDKs. | Y | The SDK only notifies the user; the user must still install them and reference them manually. | Y | NuGet pulls them automatically; the user isn't notified. |
| The mechanism integrates with Microsoft Store concepts such as app manifest and Framework ID. | Y | The SDK must pass concepts that are specific to the Store so that packaging and F5 work correctly with SDKs that are available in theStore. | N | |

| FEATURE | SDK SUPPORT | SDK NOTES | NUGET SUPPORT | NUGET NOTES |
|---|-------------|--|---------------|---|
| The mechanism integrates with the app debugging pipeline for Windows 8.x Store apps. | Y | The SDK must pass Store-specific concepts so that packaging and F5 work correctly with SDKs available in the Store. | Y | NuGet content becomes part of the project. No special F5 consideration is needed. |
| The mechanism integrates with app manifests. | Y | The SDK must pass Store-specific concepts so that packaging and F5 work correctly with SDKs available in the Store. | Y | NuGet content becomes part of the project. No special F5 consideration is needed. |
| The mechanism deploys non-reference files (for example, deploy test framework upon which to run tests of Windows 8.x Store apps). | Y | If you drop the files in the \redist folder, the files are automatically deployed. | Y | |
| The mechanism automatically adds the platform SDKs in Visual Studio IDE. | Y | If you drop the Windows 8 SDK or the Windows Phone SDK in a specific location with a specific layout, the SDK is automatically integrated with all the Visual Studio features. | N | |

| FEATURE | SDK SUPPORT | SDK NOTES | NUGET SUPPORT | NUGET NOTES |
|--|-------------|--|---------------|---|
| The mechanism supports a clean developer machine. (That is, no installation is required, and simple retrieval from source code control will work.) | N | <p>Because you reference an SDK, you must check in your solution and the SDK separately. You can check in the SDK from the two non-registry default locations from which MSBuild iterates SDKs (for details, see Creating a Software Development Kit). As an alternative, if a custom location consists of the SDKs, you can specify the following code in the project file:</p> <pre><PropertyGroup> <SDKReferenceDirectoryRoot>C:\MySDKs</SDKReferenceDirectoryRoot> </PropertyGroup></pre> <p>Then check the SDKs into that location.</p> | Y | You can check out the solution, and Visual Studio immediately recognizes and acts on the files. |
| You can join a large existing community of package authors. | N/A | The community is new. | Y | |
| You can join a large existing community of package consumers. | N/A | The community is new. | Y | |
| You can join an ecosystem of partners (custom galleries, repositories, and so forth). | N/A | The available repositories include Visual Studio Marketplace, Microsoft Download Center, and Microsoft Store. | Y | |
| The mechanism integrates with continuous-integration build servers for both package creation and consumption. | Y | The SDK must pass the checked-in location (SDKReferenceDirectoryRoot property) on command line to MSBuild. | Y | |
| The mechanism supports both stable and pre-release package versions. | Y | The SDK supports adding references to multiple versions. | Y | |

| FEATURE | SDK SUPPORT | SDK NOTES | NUGET SUPPORT | NUGET NOTES |
|---|-------------------------------|--|---------------|--|
| The mechanism supports auto-update for installed packages. | Y | If shipped as VSIX or part of Visual Studio automatic updates, SDK provides automatic notifications. | Y | |
| The mechanism contains a stand-alone .exe file for creating and consuming packages. | Y | The SDK contains MSBuild.exe. | Y | |
| Packages can be checked into version control. | Y | You can't check in anything outside the Documents node, which means that the Extension SDKs might not be checked in. The size of Extension SDK might be large. | Y | |
| You can use a PowerShell interface to create and consume packages. | Y (consumption), N (creation) | No tooling for creating an SDK. Consumption is executing MSBuild on the command line. | Y | |
| You can use a Symbol package for debugging support. | Y | If you drop .pdb files in the SDK, the files get picked up automatically. | Y | |
| The mechanism supports package manager auto-updates. | N/A | The SDK gets revised with MSBuild. | Y | |
| The mechanism supports a lightweight manifest format. | Y | SDKManifest.xml supports many attributes, but a small subset is usually necessary. | Y | |
| The mechanism is available for all Visual Studio editions. | Y | The SDK supports all Visual Studio editions. | Y | NuGet supports all Visual Studio editions. |
| The mechanism is available for all project types. | N | The SDK supports Windows 8.x Store apps starting in Visual Studio 2012. | N | You can review a list of allowed projects. |

See also

[Managing references in a project](#)

How to: Add or Remove Imported Namespaces (Visual Basic)

12/22/2017 • 2 min to read • [Edit Online](#)

Importing a namespace allows you to use elements from that namespace in your code without fully qualifying the element. For example, if you want to access the `Create` method in the `System.Messaging.MessageQueue` class, you can import the `System.Messaging` namespace and just refer to the element you need in code as `MessageQueue.Create`.

Imported namespaces are managed on the **References** page of the **Project Designer**. The imports you specify in this dialog box are passed directly to the compiler (`/imports`) and apply to all files in your project. Use the `Imports` statement to use a namespace in a single source code file.

To add an imported namespace

1. In **Solution Explorer**, double-click the **My Project** node for the project.
2. In the **Project Designer**, click the **References** tab.
3. In the **Imported Namespaces** list, select the check box for the namespace that you wish to add.

NOTE

In order to be imported, the namespace must be in a referenced component. If the namespace does not appear in the list, you will need to add a reference to the component that contains it. For more information, see [Managing references in a project](#).

To remove an imported namespace

1. In **Solution Explorer**, double-click the **My Project** node for the project.
2. In the **Project Designer**, click the **References** tab.
3. In the **Imported Namespaces** list, clear the check box for the namespace that you wish to remove.

User Imports

User imports allow you to import a specific class within a namespace rather than the entire namespace. For example, your application might have an import for the `System.Diagnostics` namespace, but the only class within that namespace that you are interested in is the `Debug` class. You can define `System.Diagnostics.Debug` as a user import, and then remove the import for `System.Diagnostics`.

If you later change your mind and decide that was really the `EventLog` class that you needed, you could enter `System.Diagnostics.EventLog` as a user import and overwrite `System.Diagnostics.Debug` using the update functionality.

To add a user import

1. In **Solution Explorer**, double-click the **My Project** node for the project.
2. In the **Project Designer**, click the **References** tab.
3. In the text box below the **Imported Namespaces** list, enter the full name for the namespace you wish to import, including the root namespace.

4. Click the **Add user import** button to add the namespace to the **Imported Namespaces** list.

NOTE

The **Add user import** button will be disabled if the namespace matches one already in the list; you cannot add an import twice.

To update a user import

1. In **Solution Explorer**, double-click the **My Project** node for the project.
2. In the **Project Designer**, click the **References** tab.
3. In the **Imported Namespaces** list, select the namespace you wish to change.
4. In the text box below the **Imported Namespaces** list, enter the name for the new namespace.
5. Click the **Update user import** button to update the namespace in the **Imported Namespaces** list.

See Also

[Managing references in a project](#)

Troubleshoot broken references

1/26/2018 • 3 min to read • [Edit Online](#)

If your application attempts to use a broken reference, an exception error is generated. The inability to find the referenced component is the primary trigger for the error, but there are several situations in which a reference can be considered broken. These instances are shown in the following list:

- The project's reference path is incorrect or incomplete.
- The file being referenced has been deleted.
- The file being referenced has been renamed.
- The network connection or authentication has failed.
- The reference is to a COM component that is not installed on the computer.

The following are remedies to these problems.

NOTE

Files in assemblies are referenced with absolute paths in the project file. Therefore, it is possible for users who work in a multideveloper environment to be missing a referenced assembly in their local environment. To avoid these errors, it is better in these cases to add project-to-project references. For more information, see [Programming with Assemblies](#).

Reference path is incorrect

If projects are shared on different computers, some references might not be found when a component is located in a different directory on each computer. References are stored under the name of the component file (for example, MyComponent). When a reference is added to a project, the folder location of the component file (for example, C:\MyComponents\) is appended to the **ReferencePath** project property.

When the project is opened, it attempts to locate these referenced component files by looking in the directories on the reference path. If the project is opened on a computer that stores the component in a different directory, such as D:\MyComponents\, the reference cannot be found and an error appears in the Task List.

To fix this problem, you can delete the broken reference and then replace it using the Add Reference dialog box. Another solution is to use the **Reference Path** item in the project's property pages and modify the folders in the list to point to the correct locations. The **Reference Path** property is persisted for each user on each computer. Therefore, modifying your reference path does not affect other users of the project.

TIP

Project-to-project references do not have these problems. For this reason, use them instead of file references, if you can.

To fix a broken project reference by correcting the reference path

1. In **Solution Explorer**, right-click your project node and click **Properties**.

The **Project Designer** appears.

2. If you are using Visual Basic, select the **References** page and click the **Reference Paths** button. In the **Reference Paths** dialog box, type the path of the folder that contains the item you want to reference in the

Folder field, and then click the **Add Folder** button.

If you are using C#, select the **Reference Paths** page. In the **Folder** field, type the path of the folder that contains the item you want to reference, and then click the **Add Folder** button.

Referenced file has been deleted

It is possible that the file being referenced has been deleted and no longer exists on the drive.

To fix a broken project reference for a file that no longer exists on your drive

- Delete the reference.
- If the reference exists in another location on your computer, read it from that location.

Referenced file has been renamed

It is possible that the file being referenced has been renamed.

To fix a broken reference for a file that has been renamed

- Delete the reference, and then add a reference to the renamed file.
- If the reference exists in another location on your computer, you have to read it in from that location.

Network connection or authentication has failed

There can be many possible causes for inaccessible files: a failed network connection or a failed authentication, for example. Each cause might have a unique means of recovery; for example, you might have to contact the local administrator for access to the required resources. However, deleting the reference and fixing the code which used it is always an option.

COM component is not installed on computer

If a user has added a reference to a COM component and a second user tries to run the code on a computer that does not have this component installed, the second user will receive an error that the reference is broken.

Installing the component on the second computer will correct the error. For more information about how to use references to COM components in your projects, see [COM Interoperability in .NET Framework Applications](#).

See also

[References Page, Project Designer \(Visual Basic\)](#)

Managing application resources (.NET)

1/12/2018 • 1 min to read • [Edit Online](#)

Resource files are files that are part of an application but are not compiled, for example icon files or audio files. Since these files are not part of the compilation process, you can change them without having to recompile your binaries. If you are planning to localize your application, you should use resource files for all the strings and other resources that need to be changed when you localize your application.

For more information about resources in .NET desktop apps, see [Resources in Desktop Apps](#).

Working with resources

In a managed code project, open the project properties window. You can open the properties window by either:

- right-clicking the project node in **Solution Explorer** and selecting **Properties**
- typing "project properties" in the **Quick Launch** window
- choosing **Alt+Enter** in the **Solution Explorer** window

Select the **Resources** tab. You can add a .resx file if your project does not contain one already, add and delete different kinds of resources, and modify existing resources.

Resources in other project types

Resources are managed differently in .NET projects than in other project types. For more information about resources in:

- Universal Windows Platform (UWP) apps, see [App resources and the Resource Management System](#)
- C++ projects, see [Working with Resource Files](#) and [How to: Create a Resource](#)

See also

[Resources in Desktop Apps \(.NET Framework\)](#)

Managing application settings (.NET)

1/23/2018 • 5 min to read • [Edit Online](#)

Application settings enable you to store application information dynamically. Settings allow you to store information on the client computer that should not be included in the application code (for example a connection string), user preferences and other information you need at runtime.

Application settings replace the dynamic properties used in earlier versions of Visual Studio.

Each application setting must have a unique name. The name can be any combination of letters, numbers, or an underscore that does not start with a number, and it cannot contain spaces. The name can be changed through the `Name` property.

Application settings can be stored as any data type that can be serialized to XML or has a `TypeConverter` that implements `ToString` / `FromString`. The most common types are `String`, `Integer`, and `Boolean`, but you can also store values as `Color`, `Object`, or as a connection string.

Application settings also contain a value. The value is set with the `Value` property and must match the data type of the setting.

In addition, application settings can be bound to a property of a form or control at design time.

There are two types of application settings, based on scope:

- Application-scoped settings can be used for information such as a URL for a Web service or a database connection string. These values are associated with the application. Therefore, users cannot change them at run time.
- User-scoped settings can be used for information such as persisting the last position of a form or a font preference. Users can change these values at run time.

You can change the type of a setting by using the `Scope` property.

The project system stores application settings in two XML files:

- an `app.config` file, which is created at design time when you create the first application setting
- a `user.config` file, which is created at run time when the user who runs the application changes the value of any user setting.

Notice that changes in user settings are not written to disk unless the application specifically calls a method to do this.

Creating application settings at design time

At design time, you can create application settings in two ways: by using the **Settings** page of the **Project Designer**, or by using the **Properties** window for a form or control, which allows you to bind a setting to a property.

When you create an application-scoped setting (for example, a database connection string, or a reference to server resources), Visual Studio saves it in `app.config` with the `<applicationSettings>` tag. (Connection strings are saved under the `<connectionStrings>` tag.)

When you create a user-scoped setting (for example, default font, home page, or window size), Visual Studio saves it in `app.config` with the `<userSettings>` tag.

IMPORTANT

When you store connection strings in app.config, you should take precautions to avoid revealing sensitive information, such as passwords or server paths, in the connection string.

If you take connection string information from an external source, such as a user supplying a user ID and password, you must be careful to ensure that the values that you use to construct your connection string do not contain additional connection string parameters that change the behavior of your connection.

Consider using the Protected Configuration feature to encrypt sensitive information in the configuration file. See [Protecting Connection Information](#) for more information.

NOTE

Because there is no configuration file model for class libraries, application settings do not apply for Class Library projects. The exception is a Visual Studio Tools for Office DLL project, which can have a configuration file.

Using customized settings files

You can add customized settings files to your project for convenient management of groups of settings. Settings that are contained in a single file are loaded and saved as a unit. Therefore, being able to store settings in separate files for frequently-used and infrequently-used groups can save time in loading and saving settings.

For example, you can add a file such as SpecialSettings.settings to your project. While your `SpecialSettings` class is not exposed in the `My` namespace, **View Code** can read the custom settings file that contains `Partial Class SpecialSettings`.

The Settings Designer first searches for the Settings.settings file that the project system creates; this is the default file that the Project Designer displays in the **Settings** tab. Settings.settings is located in the My Project folder for Visual Basic projects and in the Properties folder for Visual C# projects. The Project Designer then searches for other settings files in the project's root folder. Therefore, you should put your custom settings file there. If you add a .settings file elsewhere in your project, the Project Designer will not be able to locate it.

Accessing or changing application settings at run time in Visual Basic

In Visual Basic projects, you can access application settings at run time by using the `My.Settings` object. On the **Settings** page, click the **View code** button to view the Settings.vb file. Settings.vb defines the `Settings` class, which enables you to handle these events on the settings class: `SettingChanging`, `PropertyChanged`, `SettingsLoaded`, and `SettingsSaving`. Notice that the `Settings` class in Settings.vb is a partial class that displays only the user-owned code, not the whole generated class. For more information about accessing application settings by using the `My.Settings` object, see [Accessing Application Settings \(.NET Framework\)](#).

The values of any user-scoped settings that the user changes at run time (for example, the position of a form) are stored in a user.config file. Notice that the default values are still saved in app.config.

If you have changed any user-scoped settings during run time, for example in testing the application, and want to reset these settings to their default values, click the **Synchronize** button.

We strongly recommend that you use the `My.Settings` object and the default .settings file to access settings. This is because you can use the Settings Designer to assign properties to settings, and, additionally, user settings are automatically saved before application shutdown. However, your Visual Basic application can access settings directly. In that case you have to access the `MySettings` class and use a custom .settings file in the root of the project. You must also save the user settings before ending the application, as you would do for a C# application; this is described in the following section.

Accessing or changing application settings at run time in C#

In languages other than Visual Basic, such as C#, you must access the `Settings` class directly, as shown in the following Visual C# example.

```
Properties.Settings.Default.FirstUserSetting = "abc";
```

You must also explicitly call the `Save` method of this wrapper class in order to persist the user settings. You usually do this in the `Closing` event handler of the main form. The following C# example shows a call to the `Save` method.

```
Properties.Settings.Default.Save();
```

For general information about accessing application settings through the `Settings` class, see [Application Settings Overview \(.NET Framework\)](#). For information about iterating through the settings, see this [forum post](#).

See also

[Accessing Application Settings \(.NET Framework\)](#)

How to: Add an Application Configuration File to a C# Project

1/13/2018 • 1 min to read • [Edit Online](#)

By adding an application configuration file (app.config file) to a C# project, you can customize how the common language runtime locates and loads assembly files. For more information about application configuration files, see [How the Runtime locates assemblies \(.NET Framework\)](#).

NOTE

UWP apps don't contain an app.config file.

When you build your project, the development environment automatically copies your app.config file, changes the file name of the copy to match your executable, and then moves the copy to the **bin** directory.

To add an application configuration file to a C# project

1. On the menu bar, choose **Project > Add New Item**.

The **Add New Item** dialog box appears.

2. Expand **Installed > Visual C# Items**, and then choose the **Application Configuration File** template.

3. In the **Name** text box, enter a name, and then choose the **Add** button.

A file named app.config is added to your project.

See also

[Managing Application Settings \(.NET\)](#)

[Configuration File Schema \(.NET Framework\)](#)

[Configuring Apps \(.NET Framework\)](#)

Managing Assembly and Manifest Signing

12/22/2017 • 2 min to read • [Edit Online](#)

Strong-name signing gives a software component a globally unique identity. Strong names are used to guarantee that the assembly cannot be spoofed by someone else, and to ensure that component dependencies and configuration statements map to the correct component and component version.

A strong name consists of the assembly's identity (simple text name, version number, and culture information), plus a public key token and a digital signature.

For information about signing assemblies in Visual Basic and C# projects, see [Creating and Using Strong-Named Assemblies](#).

For information about signing assemblies in Visual C++ projects, see [Strong Name Assemblies \(Assembly Signing\) \(C++/CLI\)](#).

NOTE

Strong-name signing does not protect against reverse-engineering of the assembly. To protect against reverse-engineering, see [Dotfuscator Community Edition \(CE\)](#).

Asset Types and Signing

You can sign .NET assemblies and application manifests. These include the following:

- executables (.exe)
- application manifests (.exe.manifest)
- deployment manifests (.application)
- shared component assemblies (.dll)

You must sign the following types of asset:

1. Assemblies, if you want to deploy them to the global assembly cache (GAC).
2. ClickOnce application and deployment manifests. Visual Studio enables signing by default for these applications.
3. Primary interop assemblies, which are used for COM interoperability. The TLBIMP utility enforces strong-naming when creating a primary interop assembly from a COM type library.

In general you should not sign executables. A strongly-named component cannot reference a non-strongly-named component that is deployed with the application. Visual Studio does not sign application executables, but instead signs the application manifest, which points to the weak-named executable. You should generally avoid signing components that are private to your application, because signing can make it more difficult to manage dependencies.

How to Sign an Assembly in Visual Studio

You sign an application or component by using the **Signing** tab of the project properties window (right-click the project node in the **Solution Explorer** and select **Properties**, or type **project properties** in the **Quick Launch** window, or press ALT+ ENTER inside the **Solution Explorer** window). Select the **Signing** tab, then select the **Sign**

the assembly check box.

Specify a key file. If you choose to create a new key file, note that new key files are always created in the .pfx format. You need a name and password for the new file.

WARNING

You should always protect your key file with a password to prevent someone else from using it. You can also secure your keys by using providers or certificate stores.

You can also point to a key you have already created. For more information about creating keys, see [How to: Create a Public-Private Key Pair](#).

If you have access only to a public key, you can use delay signing to defer assigning the key. You enable delay signing by selecting the **Delay sign only** check box. A delay-signed project won't run, and you can't debug it. However, you can skip verification during development by using the [Sn.exe \(Strong Name Tool\)](#) with the `-Vr` option.

For information about signing manifests, see [How to: Sign Application and Deployment Manifests](#).

See Also

[Strong-Named Assemblies](#)

[Strong Name Assemblies \(Assembly Signing\) \(C++/CLI\)](#)

How to: Sign Application and Deployment Manifests

12/22/2017 • 4 min to read • [Edit Online](#)

If you want to publish an application by using ClickOnce deployment, the application and deployment manifests must be signed with a public/private key pair and signed using Authenticode technology. You can sign the manifests by using a certificate from the Windows certificate store or a key file.

For more information about ClickOnce deployment, see [ClickOnce Security and Deployment](#).

Signing the ClickOnce manifests is optional for .exe-based applications. For more information, see the "Generating Unsigned Manifests" section of this document.

For information about creating key files, see [How to: Create a Public-Private Key Pair](#).

NOTE

Visual Studio supports only Personal Information Exchange (PFX) key files that have the .pfx extension. However, you can select other types of certificates from the current user's Windows certificate store by clicking **Select from Store** on the **Signing** page of project properties.

To sign application and deployment manifests using a certificate

1. Go to the project properties window (right-click the project node in the **Solution Explorer** and select **Properties**, or type **project properties** in the **Quick Launch** window, or press ALT+ ENTER inside the **Solution Explorer** window). On the **Signing** tab, select the **Sign the ClickOnce manifests** check box.
2. Click the **Select from Store** button.

The **Select a Certificate** dialog box appears and displays the contents of the Windows certificate store.

TIP

If you click **Click here to view certificate properties**, the **Certificate Details** dialog box appears. This dialog box includes detailed information about the certificate, and includes additional options. You can click **certificates** to view additional Help information.

3. Select the certificate that you want to use to sign the manifests.
4. Additionally, you can specify the address of a timestamp server in the **Timestamp server URL** text box. This is a server that provides a timestamp specifying when the manifest was signed.

To sign application and deployment manifests using an existing key file

1. On the **Signing** page, select the **Sign the ClickOnce manifests** check box.
2. Click the **Select from File** button.

The **Select File** dialog box appears.

3. In the **Select File dialog** box, browse to the location of the key file (.pfx) that you want to use, and then click **Open**.

NOTE

This option supports only files that have the .pfx extension. If you have a key file or certificate in another format, store it in the Windows certificate store and select the certificate is described in the previous procedure. The selected certificate's purpose should include code signing.

The **Enter password to open file** dialog box appears. (If the .pfx file is already stored in your Windows certificate store, or is not password protected, you will not be prompted to enter a password.)

4. Enter the password to access the key file, and press ENTER.

To sign application and deployment manifests using a test certificate

1. On the **Signing** page, select the **Sign the ClickOnce manifests** check box.
2. To create a new certificate for testing, click the **Create Test Certificate** button.
3. In the **Create Test Certificate** dialog box, enter a password to help secure your test certificate.

Generating Unsigned Manifests

Signing the ClickOnce manifests is optional for .exe-based applications. The following procedures show how to generate unsigned ClickOnce manifests.

IMPORTANT

Unsigned manifests can simplify development and testing of your application. However, unsigned manifests introduce substantial security risks in a production environment. Only consider using unsigned manifests if your ClickOnce application runs on computers within an intranet that is completely isolated from the internet or other sources of malicious code.

By default, ClickOnce automatically generates signed manifests unless one or more files are specifically excluded from the generated hash. In other words, publishing the application results in signed manifests if all files are included in the hash, even when the **Sign the ClickOnce manifests** check box is cleared.

To generate unsigned manifests and include all files in the generated hash

1. To generate unsigned manifests that include all files in the hash, you must first publish the application together with signed manifests. Therefore, first sign the ClickOnce manifests by following one of the previous procedures, and then publish the application.
2. On the **Signing** page, clear the **Sign the ClickOnce manifests** check box.
3. Reset the publish version so that only one version of your application is available. By default, Visual Studio automatically increments the revision number of the publish version every time that you publish an application. For more information, see [How to: Set the ClickOnce Publish Version](#).
4. Publish the application.

To generate unsigned manifests and exclude one or more files from the generated hash

1. On the **Signing** page, clear the **Sign the ClickOnce manifests** check box.
2. Open the **Application Files** dialog box and set the **Hash to Exclude** for the files that you want to exclude from the generated hash.

NOTE

Excluding a file from the hash configures ClickOnce to disable automatic signing of the manifests, so you do not need to first publish with signed manifests as shown in the previous procedure.

3. Publish the application.

See Also

[Strong-Named Assemblies](#)

[How to: Create a Public-Private Key Pair](#)

[Signing Page, Project Designer](#)

[ClickOnce Security and Deployment](#)

How to: Specify an Application Icon (Visual Basic, C#)

1/26/2018 • 1 min to read • [Edit Online](#)

The **Icon** property for a project specifies the icon file (.ico) that will be displayed for the compiled application in File Explorer and in the Windows taskbar.

The **Icon** property can be accessed in the **Application** pane of the **Project Designer**; it contains a list of icons that have been added to a project either as resources or as content files.

NOTE

After you set the icon property for an application, you might also set the **Icon** property of each **Window** or **Form** in the application. For information about window icons for Windows Presentation Foundation (WPF) standalone applications, see [Icon](#) property.

To specify an application icon

1. In **Solution Explorer**, choose a project node (not the **Solution** node).
2. On the menu bar, choose **Project** > **Properties**.
3. When the **Project Designer** appears, choose the **Application** tab.
4. **(Visual Basic)**—In the **Icon** list, choose an icon (.ico) file.

C#—Near the **Icon** list, choose the **<Browse...>** button, and then browse to the location of the icon file that you want.

See also

[Application Page, Project Designer \(Visual Basic\)](#)

[Application Page, Project Designer \(C#\)](#)

Visual Studio multi-targeting overview

3/12/2018 • 3 min to read • [Edit Online](#)

In Visual Studio, you can specify the version or profile of the .NET Framework that you want your project to target. For an application to run on another computer, the Framework version that the application targets must be compatible with the Framework version that is installed on the computer.

You can also create a solution that contains projects that target different versions of the framework. Framework targeting helps guarantee that the application uses only functionality that is available in the specified version of the framework.

TIP

You can also target applications for different platforms. For more information, see [Multitargeting](#).

Framework targeting features

Framework targeting includes the following features:

- When you open a project that targets an earlier version of the .NET Framework, Visual Studio can automatically upgrade it or leave the target as-is.
- When you create a project, you can specify the version of the .NET Framework that you want to target.
- You can change the version of the .NET Framework that an existing project targets.
- You can target a different version of the .NET Framework in each of several projects in the same solution.
- When you change the version of the .NET Framework that a project targets, Visual Studio makes any required changes to references and configuration files.

When you work on a project that targets an earlier version of the .NET Framework, Visual Studio dynamically changes the development environment, as follows:

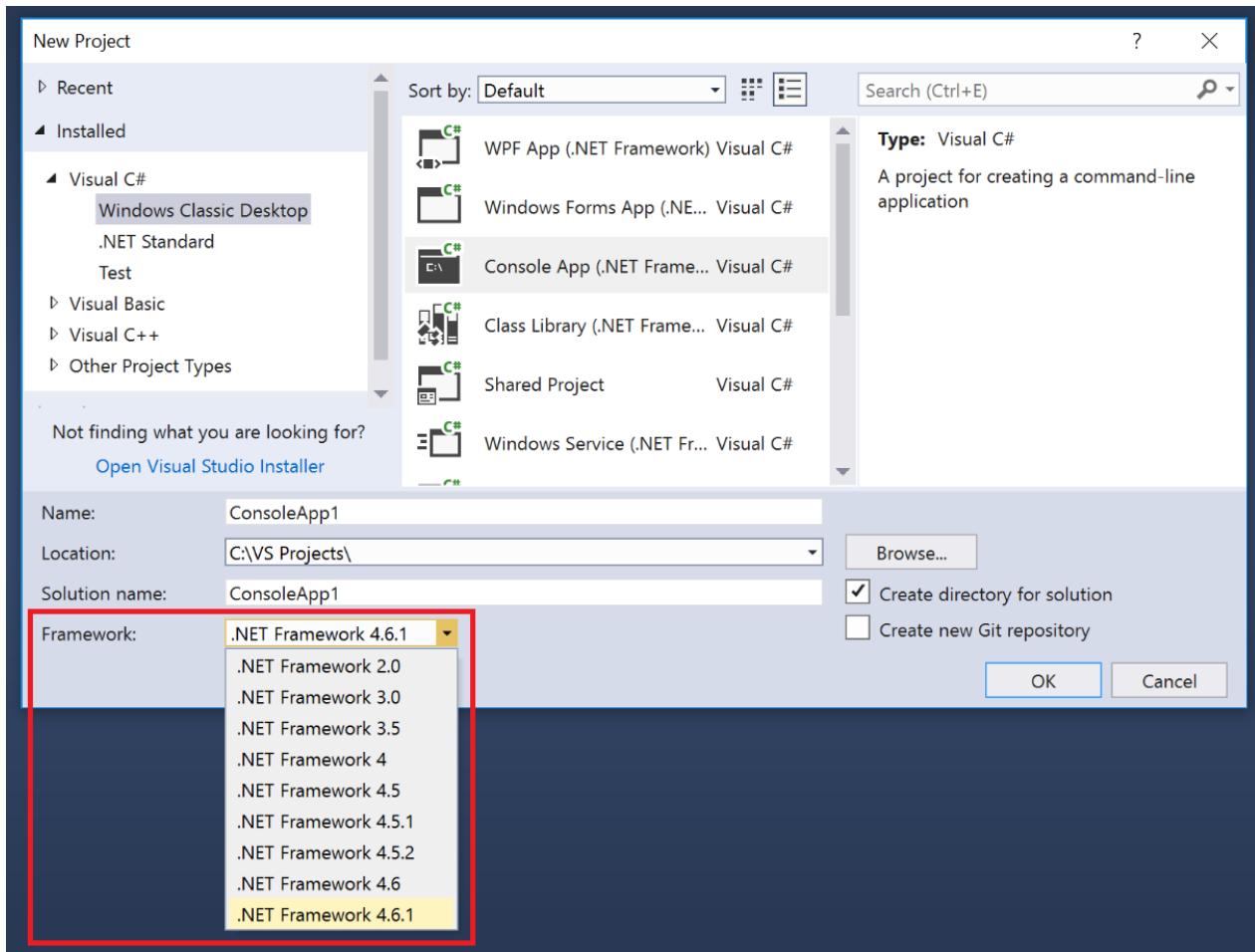
- It filters items in the **Add New Item** dialog box, the **Add New Reference** dialog box, and the **Add Service Reference** dialog box to omit choices that are not available in the targeted version.
- It filters custom controls in the **Toolbox** to remove those that are not available in the targeted version and to show the only the most up-to-date controls when multiple controls are available.
- It filters IntelliSense to omit language features that are not available in the targeted version.
- It filters properties in the **Properties** window to omit those that are not available in the targeted version.
- It filters menu options to omit options that are not available in the targeted version.
- For builds, it uses the version of the compiler and the compiler options that are appropriate for the targeted version.

NOTE

Framework targeting does not guarantee that your application will run correctly. You must test your application to make sure it runs against the targeted version. You cannot target framework versions that are earlier than the .NET Framework 2.0.

Selecting a target framework version

When you create a project, select the target .NET Framework version in the **New Project** dialog box. The list of available frameworks includes the installed framework versions that are applicable to the selected template type. For template types that don't require .NET Framework, for example .NET Core templates, the **Framework** drop-down list is hidden.



In an existing project, you can change the target .NET Framework version in the project properties dialog box. For more information, see [How to: Target a Version of the .NET Framework](#).

Resolving system and user assembly references

To target a .NET Framework version, you must first install the appropriate assembly references. You can download developer packs for different versions of the .NET Framework on the [.NET downloads](#) page.

The **Add Reference** dialog box disables system assemblies that do not pertain to the target .NET Framework version so that they cannot be added to a project inadvertently. (System assemblies are .dll files that are included in a .NET Framework version.) References that belong to a framework version that is later than the targeted version will not resolve, and controls that depend on such a reference cannot be added. If you want to enable such a reference, reset the .NET Framework target of the project to one that includes the reference. For more information, see [How to: Target a Version of the .NET Framework](#).

For more information about assembly references, see [Resolving Assemblies at Design Time](#).

Enabling LINQ

When you target the .NET Framework 3.5 or later, a reference to System.Core and a project-level import for System.Linq (in Visual Basic only) are added automatically. If you want to use LINQ features, you must also turn Option Infer on (in Visual Basic only). The reference and import are removed automatically if you change the target to an earlier .NET Framework version. For more information, see [Working with LINQ](#).

See also

- [Multitargeting \(MSBuild\)](#)
- [How to: Modify the Target Framework and Platform Toolset \(C++\)](#)

How to: Target a version of the .NET Framework

3/5/2018 • 2 min to read • [Edit Online](#)

This document describes how to target a version of the .NET Framework when you create a project, and how to change the targeted version in an existing Visual Basic, C#, or Visual F# project.

IMPORTANT

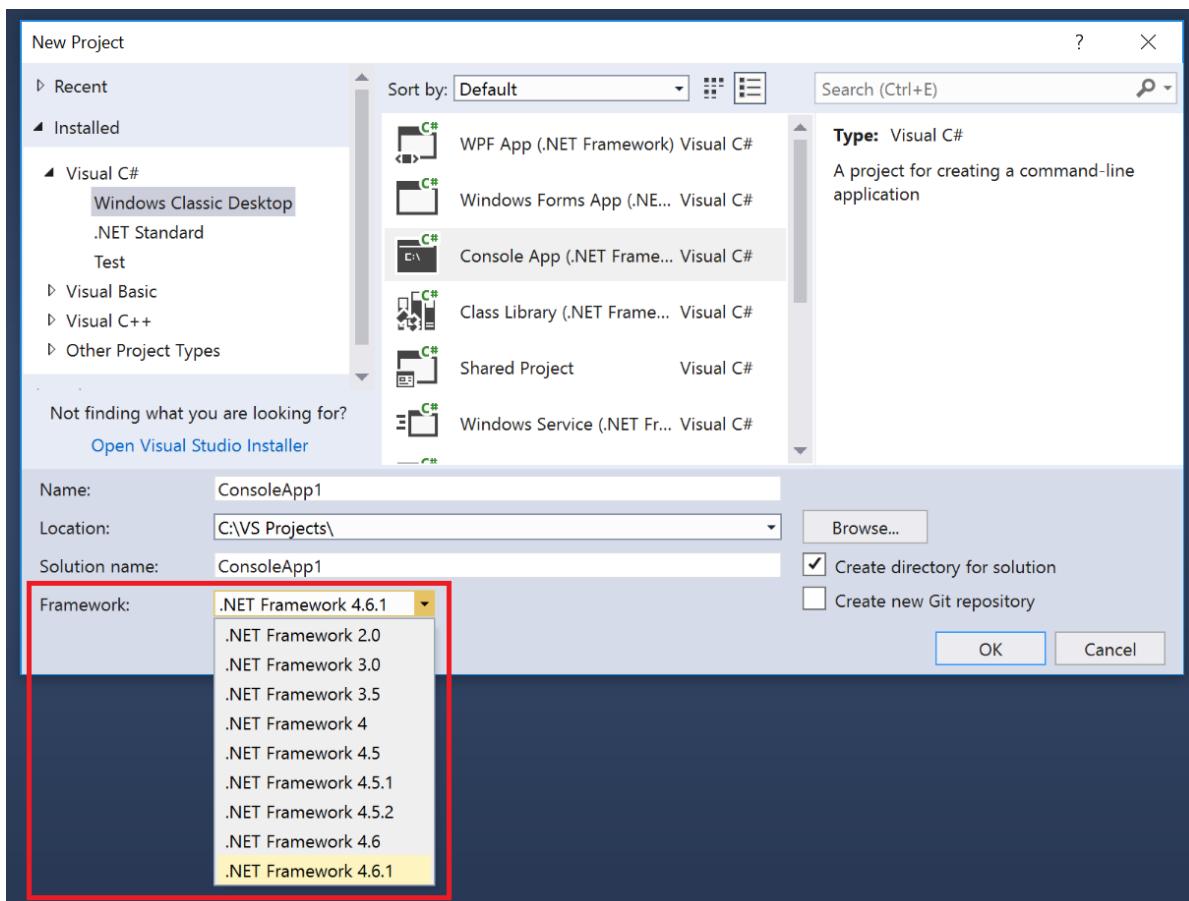
For information about how to change the target version for C++ projects, see [How to: Modify the Target Framework and Platform Toolset](#).

To target a version when you create a project

When you create a project, the available .NET Framework versions depend on which versions are installed, and the selected template in the **New Project** dialog box.

1. On the menu bar, choose **File > New > Project....**
2. In the list of installed templates, choose the type of project that you want to create, and enter a name for the project.
3. From the **Framework** drop-down list at the bottom of the **New Project** dialog box, choose the version of the .NET Framework that you want your project to target.

The list of frameworks shows only those versions that are applicable to the template that you chose. Some project types, such as .NET Core, do not require .NET Framework. In such instances, the **Framework** drop-down list is hidden.



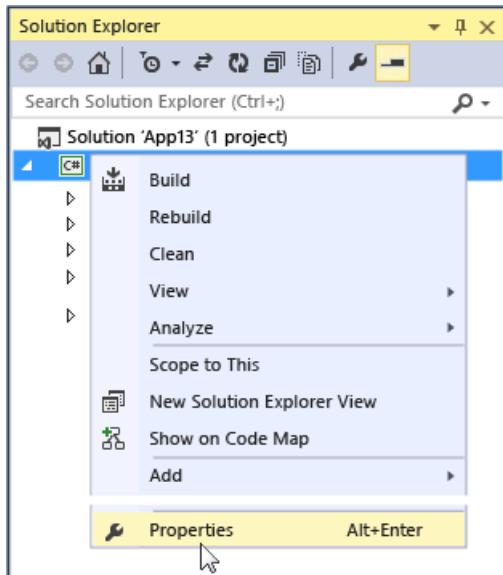
4. Choose the **OK** button.

To change the targeted version

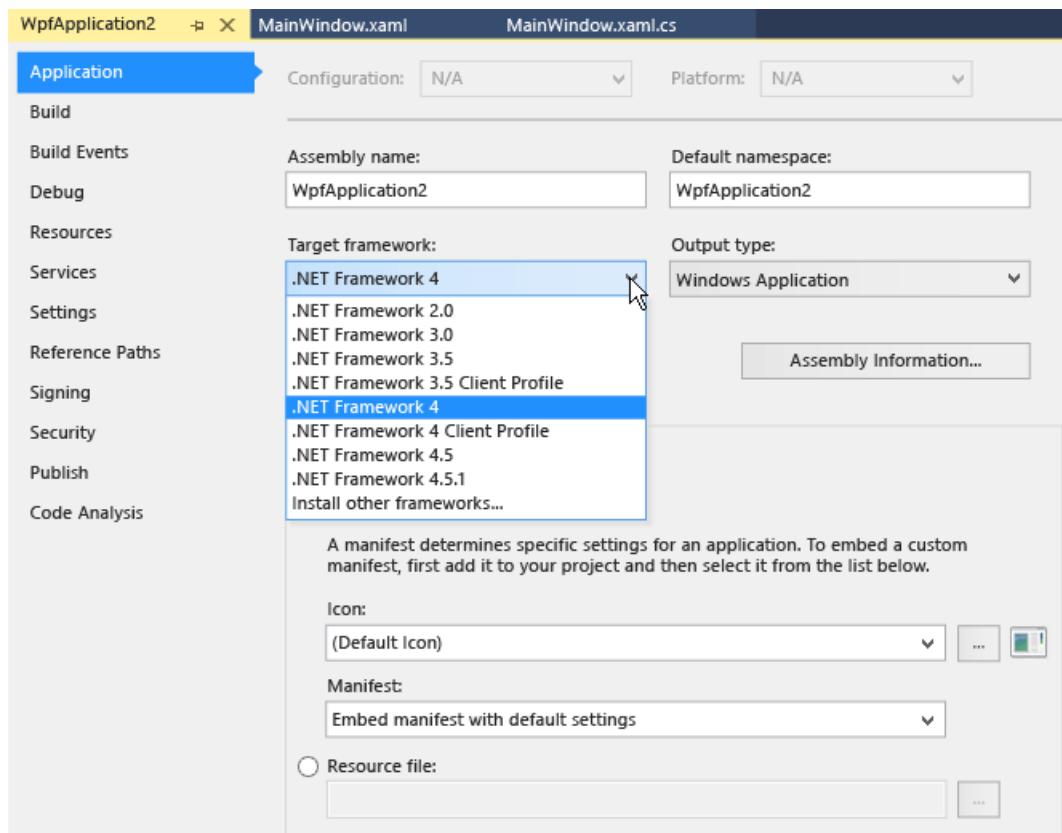
You can change the targeted version of the .NET Framework in a Visual Basic, C#, or Visual F# project by following this procedure.

For information about how to change the target version for C++ projects, see [How to: Modify the Target Framework and Platform Toolset](#).

1. In **Solution Explorer**, open the shortcut menu for the project that you want to change, and then choose **Properties**.



2. In the left column of the properties window, choose the **Application** tab.



NOTE

After you create a UWP app, you can't change the targeted version of either Windows or the .NET Framework.

3. In the **Target Framework** list, choose the version that you want.
4. In the verification dialog box that appears, choose the **Yes** button.

The project unloads. When it reloads, it targets the .NET Framework version that you just chose.

NOTE

If your code contains references to a different version of the .NET Framework than the one that you targeted, error messages may appear when you compile or run the code. To resolve these errors, you must modify the references. See [Troubleshooting .NET Framework Targeting Errors](#).

See also

- [Visual Studio Multi-Targeting Overview](#)
- [Troubleshooting .NET Framework Targeting Errors](#)
- [Application Page, Project Designer \(C#\)](#)
- [Application Page, Project Designer \(Visual Basic\)](#)
- [How to: Modify the Target Framework and Platform Toolset \(C++\)](#)

Project and item templates

1/5/2018 • 2 min to read • [Edit Online](#)

Project and item templates provide reusable stubs that give users some basic code and structure, that they can customize for their own purposes.

Visual Studio templates

A number of predefined project and item templates are installed with Visual Studio. For example, the Visual Basic and C# **Windows Forms App** and **Class Library** templates that are shown in the **New Project** dialog box are project templates. Item templates are shown in the **Add New Item** dialog box, and include items such as code files, XML files, HTML pages, and Style Sheets.

These templates provide a starting point for users to begin creating projects, or to expand existing projects. Project templates provide the files that are required for a particular project type, include standard assembly references, and set default project properties and compiler options. Item templates can range in complexity from a single empty file that has a certain file extension, to a multi-file item that contains, for example, source code files that have stub code, designer information files, and embedded resources.

In addition to the installed templates in the **New Project** and **Add New Item** dialog boxes, you can author your own templates, or download and use templates created by the community. For more information, see [How to: Create project templates](#) and [How to: Create item templates](#).

Contents of a template

All project and item templates, whether installed with Visual Studio or created by you, function using the same principles and have similar contents. All templates contain the following items:

- The files to be created when the template is used. This includes source code files, embedded resources, project files, and so on.
- One .vstemplate file. This file contains metadata that provides the information needed to display the template in the **New Project** and **Add New Item** dialog boxes, and to create a project or item from the template. For more information about .vstemplate files, see [Template parameters](#).

When these files are compressed into a .zip file and put in the correct folder, Visual Studio automatically displays them in the following places:

- Project templates appear in the **New Project** dialog box.
- Item templates appear in the **Add New Item** dialog box.

For more information about template folders, see [How to: Locate and organize templates](#).

Starter Kits

Starter Kits are enhanced templates that can be shared with other members of the community. A Starter Kit includes code samples that compile, documentation, and other resources to help users learn new tools and programming techniques while they build useful, real-world applications. The basic contents and procedures for Starter Kits are identical to those for templates. For more information, see [How to: Create Starter Kits](#).

See also

[How to: Create project templates](#)

[How to: Create item templates](#)

[Template parameters](#)

[Customizing templates](#)

[NuGet Packages in Visual Studio templates](#)

How to: Create project templates

1/5/2018 • 2 min to read • [Edit Online](#)

This topic shows you how to create a template using the **Export Template Wizard**, which packages your template in a .zip file.

To create a user project template by using the Export Template Wizard

1. Create a project.

NOTE

Use only valid identifier characters when naming a project that will be the source for a template. Otherwise, compilation errors can occur in projects that are created from the template. For more information about valid identifier characters, see [Declared element names \(Visual Basic\)](#) or [Identifiers \(C++\)](#). Alternatively, you can use [template parameters](#) to use "safe" names for classes and namespaces.

2. Edit the project until it is ready to be exported as a template. For example, you might want to edit code files to indicate where parameter replacement should take place. See [How to: Substitute parameters in a template](#).

3. On the **Project** menu, choose **Export Template....**

The **Export Template Wizard** opens.

4. On the **Choose Template Type** page, select **Project Template**. Select the project you want to export to a template, and then choose **Next**.
5. On the **Select Template Options** page, enter a name, and optional description, icon, and preview image for your template. These items will appear in the **New Project** dialog box. Choose **Finish**.

The project is exported into a .zip file and placed in the specified output location, and, if selected, imported into Visual Studio.

NOTE

To find your template in the **New Project** dialog box, expand **Installed** and then expand the category that corresponds to the `ProjectType` element in the .vstemplate file. For example, a .vstemplate file that contains `<ProjectType>CSharp</ProjectType>` appears under **Installed > Visual C#**, by default. You can organize your template into a subdirectory of the project type just by creating a folder in that directory and placing your template's .zip file in it. For more information, see [How to: locate and organize templates](#).

Other ways to create project templates

You can create project templates manually by gathering the files that constitute the project into a folder, and then creating a .vstemplate XML file with the appropriate metadata. For more information, see [How to: Manually create web templates](#).

If you have the Visual Studio SDK installed, you can wrap the finished template in a VSIX file for deployment by using the **VSIX Project** template. For more information, see [Getting started with the VSIX project template](#).

See also

[Creating Project and Item Templates](#)

[How to: Create Item Templates](#)

[Getting started with the VSIX project template](#)

How to: Create multi-project templates

1/5/2018 • 3 min to read • [Edit Online](#)

Multi-project templates act as containers for two or more projects. When you create a project that is based on a multi-project template from the **New Project** dialog box, every project in the template is added to the solution.

A multi-project template contains two or more project templates, and a root template of type `ProjectGroup`.

Multi-project templates behave differently than single project templates. They have the following unique characteristics:

- Individual projects in a multi-project template cannot be assigned names in the **New Project** dialog box. Instead, use the `ProjectName` attribute on the `ProjectTemplateLink` element in the .vstemplate file to specify a name for each project.
- Multi-project templates can contain projects for different languages, but the entire template itself can only be put in one category. Specify the template category in the `ProjectType` element of the .vstemplate file.

A multi-project template must include the following items, compressed into a .zip file:

- A root .vstemplate file for the entire multi-project template. This root .vstemplate file contains metadata that the **New Project** dialog box displays, and specifies where to find the .vstemplate files for the projects in the template. This file must be located at the root of the .zip file.
- Two or more folders that contain the files that are required for a complete project template. This includes all code files for the project, and also a .vstemplate file for the project.

For example, a multi-project template .zip file that has two projects could have the following files and directories:

- MultiProjectTemplate.vstemplate
- \Project1\Project1.vstemplate
- \Project1\Project1.vbproj
- \Project1\Class.vb
- \Project2\Project2.vstemplate
- \Project2\Project2.vbproj
- \Project2\Class.vb

The root .vstemplate file for a multi-project template differs from a single-project template in the following ways:

- The `Type` attribute of the `VSTemplate` element has the value `ProjectGroup` instead of `Project`. For example:

```
<VSTemplate Version="2.0.0" Type="ProjectGroup"
  xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">
```

- The `TemplateContent` element contains a `ProjectCollection` element that has one or more `ProjectTemplateLink` elements that define the paths to the .vstemplate files of the included projects. For example:

```
<TemplateContent>
  <ProjectCollection>
    <ProjectTemplateLink>
      Project1\Project1.vstemplate
    </ProjectTemplateLink>
    <ProjectTemplateLink>
      Project2\Project2.vstemplate
    </ProjectTemplateLink>
  </ProjectCollection>
</TemplateContent>
```

To create a multi-project template from an existing solution

1. Create a solution and add two or more projects.
2. Customize the projects until they are ready to be exported to a template.
3. On the **Project** menu, choose **Export Template....**

The **Export Template Wizard** opens.

4. On the **Choose Template Type** page, select **Project Template**. Select the project you want to export to a template, and then choose **Next**.
5. On the **Select Template Options** page, enter a name and optional description, icon, and preview image for your template. Choose **Finish**.

The project is exported into a .zip file and placed in the specified output location.

NOTE

Each project must be exported to a template separately, so repeat the preceding steps for each project in the solution.

6. Create a directory for your template, with a subdirectory for each project.
7. Extract the contents of each project's .zip file into the corresponding subdirectory that you just created.
8. In the base directory, create an XML file with a **.vstemplate** file extension. This file contains the metadata for the multi-project template. See the example that follows for the structure of the file. Be sure to specify the relative path to each project's .vstemplate file.
9. Select the base directory, and from the right-click or context menu, choose **Send to > Compressed (zipped) folder**.

The files and folders are compressed into a .zip file.

10. Copy the .zip file into the user project template directory. By default, this directory is %USERPROFILE%\Documents\Visual Studio <version>\Templates\ProjectTemplates.
11. In Visual Studio, open the **New Project** dialog box and verify that your template appears.

Two-project example

This example shows a basic multi-project root .vstemplate file. In this example, the template contains two projects, **My Windows Application** and **My Class Library**. The **ProjectName** attribute on the **ProjectTemplateLink** element specifies the name that is given to the project.

TIP

If the `ProjectName` attribute is not specified, the name of the `.vstemplate` file is used as the project name.

```
<VSTemplate Version="2.0.0" Type="ProjectGroup"
  xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">
  <TemplateData>
    <Name>Multi-Project Template Sample</Name>
    <Description>An example of a multi-project template</Description>
    <Icon>Icon.ico</Icon>
    <ProjectType>VisualBasic</ProjectType>
  </TemplateData>
  <TemplateContent>
    <ProjectCollection>
      <ProjectTemplateLink ProjectName="My Windows Application">
        WindowsApp\MyTemplate.vstemplate
      </ProjectTemplateLink>
      <ProjectTemplateLink ProjectName="My Class Library">
        ClassLib\MyTemplate.vstemplate
      </ProjectTemplateLink>
    </ProjectCollection>
  </TemplateContent>
</VSTemplate>
```

Example with solution folders

This example uses the `SolutionFolder` element to divide the projects into two groups, `Math Classes` and `Graphics Classes`. The template contains four projects, two of which are placed in each solution folder.

```
<VSTemplate Version="2.0.0" Type="ProjectGroup"
  xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">
  <TemplateData>
    <Name>Multi-Project Template Sample</Name>
    <Description>An example of a multi-project template</Description>
    <Icon>Icon.ico</Icon>
    <ProjectType>VisualBasic</ProjectType>
  </TemplateData>
  <TemplateContent>
    <ProjectCollection>
      <SolutionFolder Name="Math Classes">
        <ProjectTemplateLink ProjectName="MathClassLib1">
          MathClassLib1\MyTemplate.vstemplate
        </ProjectTemplateLink>
        <ProjectTemplateLink ProjectName="MathClassLib2">
          MathClassLib2\MyTemplate.vstemplate
        </ProjectTemplateLink>
      </SolutionFolder>
      <SolutionFolder Name="Graphics Classes">
        <ProjectTemplateLink ProjectName="GraphicsClassLib1">
          GraphicsClassLib1\MyTemplate.vstemplate
        </ProjectTemplateLink>
        <ProjectTemplateLink ProjectName="GraphicsClassLib2">
          GraphicsClassLib2\MyTemplate.vstemplate
        </ProjectTemplateLink>
      </SolutionFolder>
    </ProjectCollection>
  </TemplateContent>
</VSTemplate>
```

See also

[Creating project and item templates](#)

[How to: Create project templates](#)

[Visual Studio template schema reference \(extensibility\)](#)

[SolutionFolder element \(Visual Studio templates\)](#)

[ProjectTemplateLink element \(Visual Studio templates\)](#)

How to: Create item templates

1/12/2018 • 3 min to read • [Edit Online](#)

This topic shows you how to create an item template by using the **Export Template Wizard**. If your template will consist of multiple files, see [How to: Create Multi-file Item Templates](#).

To add a user item template to the Add New Item dialog box

1. Create or open a project in Visual Studio.
2. Add an item to the project, and modify it if you want to.
3. Modify the code file to indicate where parameter replacement should take place. For more information, see [How to: Substitute parameters in a template](#).
4. On the **Project** menu, choose **Export Template...**
5. On the **Choose Template Type** page, choose **Item Template**, select the project that contains the item, and then choose **Next**.
6. On the **Select Item To Export** page, choose the item you want to create a template for, and then choose **Next**.
7. On the **Select Item References** page, select the assembly references to include in the template, and then choose **Next**.
8. On the **Select Template Options** page, enter the template name and optional description, icon image and preview image, and then choose **Finish**.

The files for the template are added to a .zip file and copied to the directory you specified in the wizard. The default location is %USERPROFILE%\Documents\Visual Studio <version>\My Exported Templates.

9. If you did not select the option **Automatically import the template into Visual Studio** in the **Export Template Wizard**, locate the exported template and copy it to the user item template directory. The default location is %USERPROFILE%\Documents\Visual Studio <version>\Templates\ItemTemplates.
10. Close Visual Studio and then reopen it.
11. Create a new project, or open an existing project, and then choose **Project > Add New Item...** or press **Ctrl + Shift + A**.

The item template appears in the **Add New Item** dialog box. If you added a description in the **Export Template Wizard**, the description appears on the right side of the dialog box.

To enable the item template to be used in a Universal Windows App project

The wizard does much of the work to create a basic template, but in many cases you need to manually modify the .vstemplate file after you have exported the template. For example, if you want the item to appear in the **Add New Item** dialog for a Universal Windows App project, you have to perform a few extra steps.

1. Follow the steps in the previous section to export an item template.
2. Extract the .zip file that was created, and open the .vstemplate file in Visual Studio.

3. For a C# Universal Windows project, add the following XML inside the `<TemplateData>` element:

```
<TemplateID>Microsoft.CSharp.Class</TemplateID>
```

4. In Visual Studio, save the .vstemplate file and close it.

5. Copy and paste the .vstemplate file back to the .zip file.

If the **Copy File** dialog box appears, choose the **Copy and Replace** option.

You can now add an item based on this template to a Universal Windows project from the **Add New Item** dialog box.

To enable templates for specific project subtypes

You can specify that your template should only appear for only certain project subtypes, such as Windows, Office, Database, or Web.

1. Locate the `ProjectType` element in the .vstemplate file for the item template.

2. Add a `ProjectSubType` element immediately after the `ProjectType` element.

3. Set the text value of the element to one of the following values:

- Windows
- Office
- Database
- Web

For example: `<ProjectSubType>Database</ProjectSubType>`.

The following example shows an item template for **Office** projects.

```
<VSTemplate Version="2.0.0" Type="Item" Version="2.0.0">
  <TemplateData>
    <Name>Class</Name>
    <Description>An empty class file</Description>
    <Icon>Class.ico</Icon>
    <ProjectType>CSharp</ProjectType>
    <ProjectSubType>Office</ProjectSubType>
    <DefaultName>Class.cs</DefaultName>
  </TemplateData>
  <TemplateContent>
    <ProjectItem>Class1.cs</ProjectItem>
  </TemplateContent>
</VSTemplate>
```

To manually create an item template without using the Export Template Wizard

In some cases you may want to create an item template manually, from scratch.

1. Create a project and project item.

2. Modify the project item until it is ready to be saved as a template.

3. Modify the code file to indicate where parameter replacement should occur, if anywhere. For more information about parameter replacement, see [How to: Substitute Parameters in a Template](#).

4. Create an XML file and save it with a .vstemplate file extension in the same directory as your project item file.
5. Edit the .vstemplate XML file to provide item template metadata. For more information, see [Template schema reference \(Extensibility\)](#) and the example in the previous section.
6. Save the .vstemplate file and close it.
7. In Windows Explorer, select the files you want to include in your template, right-click the selection, and choose **Send to > Compressed (zipped) folder**. The files that you selected are compressed into a .zip file.
8. Copy the .zip file and paste it in the user item template location. In Visual Studio 2017, the default directory is %USERPROFILE%\Documents\Visual Studio 2017\Templates\ItemTemplates. For more information, see [How to: Locate and Organize Project and Item Templates](#).

See also

[Creating Project and Item Templates](#)

[How to: Create Multi-file Item Templates](#)

[Visual Studio Template Schema Reference \(Extensibility\)](#)

How to: Create multi-file item templates

1/5/2018 • 2 min to read • [Edit Online](#)

Item templates may only specify one item, but sometimes the item is made up of multiple files. For example, a Windows Forms item template requires the following three files:

- A file that contains the code for the form
- A file that contains the designer information for the form
- A file that contains the embedded resources for the form

Multi-file item templates require parameters to ensure that the correct file extensions are used when the item is created. If you create a multi-file item template by using the **Export Template Wizard**, these parameters are automatically generated, and no further editing is required.

To create a multi-file item template by using the Export Template Wizard

You can create a multi-file item template in the same manner as you would a single-file item template. See [How to: Create item templates](#). On the **Select Item To Export** page of the wizard, select the file that has dependent files (for example, a Windows Forms form file). The wizard automatically includes any dependent files, such as designer and resource files, in the template.

To manually create a multi-file item template

1. Create the item template as you would manually create a single-file item template, but include each file that constitutes the multi-file item.
2. In the .vstemplate XML file, add a `ProjectItem` element for each individual file, and add a `TargetFileName` attribute to this element. Set the value of the `TargetFileName` attribute to `$fileinputname$.FileExtension`, where `FileExtension` is the file extension of the file that is being included in the template. For example:

```
<ProjectItem TargetFileName="$fileinputname$.vb">
  Form1.vb
</ProjectItem>
<ProjectItem TargetFileName="$fileinputname$.Designer.vb">
  Form1.Designer.vb
</ProjectItem>
<ProjectItem TargetFileName="$fileinputname$.resx">
  Form1.resx
</ProjectItem>
```

NOTE

When an item derived from this template is added to a project, the file names will derive from the name that the user enters in the **Add New Item** dialog box.

3. Select the files to be included in your template, right-click the selection, and choose **Send to > Compressed (zipped) folder**.

The files that you selected are compressed into a .zip file.

4. Copy the .zip file to the user item template location. By default, the directory is %USERPROFILE%\Documents\Visual Studio <Version>\Templates\ItemTemplates. For more information, see [How to: Locate and Organize Templates](#).
5. Close Visual Studio and then reopen it.
6. Create a new project, or open an existing project, and then choose **Project > Add New Item...** or press **Ctrl + Shift + A**.

The multi-file item template appears in the **Add New Item** dialog box.

Example

The following example shows a Windows Forms template. When an item is created based on this template, the names of the three files created will match the name entered in the **Add New Item** dialog box.

```
<VSTemplate Version="2.0.0" Type="Item"
  xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">
  <TemplateData>
    <Name>Multi-file Item Template</Name>
    <Icon>Icon.ico</Icon>
    <Description>An example of a multi-file item template</Description>
    <ProjectType>VisualBasic</ProjectType>
  </TemplateData>
  <TemplateContent>
    <ProjectItem TargetFileName="$fileinputname$.vb" SubType="Form">
      Form1.vb
    </ProjectItem>
    <ProjectItem TargetFileName="$fileinputname$.Designer.vb">
      Form1.Designer.vb
    </ProjectItem>
    <ProjectItem TargetFileName="$fileinputname$.resx">
      Form1.resx
    </ProjectItem>
  </TemplateContent>
</VSTemplate>
```

See also

[Creating Project and Item Templates](#)

[How to: Create Item Templates](#)

[Template Parameters](#)

[How to: Substitute Parameters in a Template](#)

How to: Manually create Web templates

1/5/2018 • 1 min to read • [Edit Online](#)

Creating a Web template is different than creating other kinds of templates. Because Web project templates appear in the **Add New Web Site** dialog box, and Web project items are categorized by programming language, the .vstemplate file must specify the template as a Web template and identify the programming language.

NOTE

Web templates must contain an empty .webproj file, and it must be referenced in the .vstemplate file in the `File` attribute of the `Project` element. Although Web projects do not require a *.proj project file, it's necessary to create this stub file for the Web template to function correctly.

To manually create a Web template

1. Create a Web project.
2. Modify or delete the files in the project, or add new files to the project.
3. Create an XML file and save it with a .vstemplate file name extension, in the same directory as your project.
Do not add it to the project in Visual Studio.
4. Edit the .vstemplate XML file to provide project template metadata. For more information, see the [example that follows](#).
5. Locate the `ProjectType` element in the .vstemplate file, and set the text value to `Web`.
6. Following the `ProjectType` element, add a `ProjectSubType` element and set the text value to the programming language of the template. The programming language can be one of the following values:
 - CSharp
 - VisualBasic

For example:

```
<TemplateData>
  ...
  <ProjectType>Web</ProjectType>
  <ProjectSubType>CSharp</ProjectSubType>
  ...
</TemplateData>
```

7. Select the files in your template (this includes the .vstemplate file), right-click the selection, and choose **Send to** > **Compressed (zipped) folder**. The files are compressed into a .zip file.
8. Put the .zip template file in the Visual Studio project template directory. By default, this directory is %USERPROFILE%\Documents\Visual Studio <Version>\ProjectTemplates.

Example

The following example shows a basic .vstemplate file for a Web project template:

```
<VSTemplate Version="2.0.0" Type="Project"
  xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">>
  <TemplateData>
    <Name>MyWebProjecStarterKit</Name>
    <Description>A simple Web template</Description>
    <Icon>icon.ico</Icon>
    <ProjectType>Web</ProjectType>
    <ProjectSubType>CSharp</ProjectSubType>
    <DefaultName>WebSite</DefaultName>
  </TemplateData>
  <TemplateContent>
    <Project File="WebApplication.webproj">
      <ProjectItem>icon.ico</ProjectItem>
      <ProjectItem OpenInEditor="true">Default.aspx</ProjectItem>
      <ProjectItem>Default.aspx.cs</ProjectItem>
    </Project>
  </TemplateContent>
</VSTemplate>
```

See also

[Creating Project and Item Templates](#)

[Visual Studio Template Schema Reference \(Extensibility\)](#)

How to: Troubleshoot templates

1/13/2018 • 1 min to read • [Edit Online](#)

If a template fails to load in the development environment, there are several ways to locate the problem.

Validate the .vstemplate file

If the .vstemplate file in a template does not adhere to the Visual Studio template schema, the template may not appear in the **New Project** dialog box.

To validate the .vstemplate file

1. Locate the .zip file that contains the template.
2. Extract the .zip file.
3. On the **File** menu in Visual Studio, choose **Open > File**.
4. Select the .vstemplate file for the template, and choose **Open**.
5. Verify that the XML of the .vstemplate file adheres to the template schema. For more information on the .vstemplate schema, see [Template schema reference](#).

NOTE

To get IntelliSense support while authoring the .vstemplate file, add a `xmlns` attribute to the `VSTemplate` element, and assign it a value of <http://schemas.microsoft.com/developer/vstemplate/2005>.

6. Save and close the .vstemplate file.
7. Select the files included in your template, right-click, and choose **Send to > Compressed (zipped) folder**.
The files that you selected are compressed into a .zip file.
8. Place the new .zip file in the same directory as the old .zip file.
9. Delete the extracted template files and the old template .zip file.

Enable diagnostic logging

You can enable diagnostic logging for template discovery by following the steps in [Troubleshooting template discovery \(Extensibility\)](#).

See also

[Troubleshooting template discovery \(Extensibility\)](#)

[Customizing templates](#)

[Creating project and item templates](#)

[Template schema reference](#)

How to: Locate and organize project and item templates

1/13/2018 • 3 min to read • [Edit Online](#)

Template files must be placed in a location that Visual Studio recognizes for the templates to appear in the **New Project** and **Add New Item** dialog boxes. You can also create custom subcategories in the user template location, and the categories are shown in the **New Project** and **Add New Item** dialog boxes.

Locate templates

Installed templates and user templates are stored in two different locations.

User templates

If you add a compressed (.zip) file that includes a .vstemplate file to the user template directory, the template appears in the **New Project** or **Add New Item** dialog box. By default, user templates are located in:

- %USERPROFILE%\Documents\Visual Studio <Version>\Templates\ProjectTemplates
- %USERPROFILE%\Documents\Visual Studio <Version>\Templates\ItemTemplates

For example, the following directory contains user project templates for C#:

C:\Users\UserName\Documents\Visual Studio 2017\Templates\ProjectTemplates\Visual C#\

TIP

You can set the location for user templates in **Tools > Options > Projects and Solutions > Locations**.

Installed templates

By default, templates installed with Visual Studio are located in:

- \VisualStudio\InstallationDirectory\Common7\IDE\ItemTemplates\Programming Language\Locale ID
- \VisualStudio\InstallationDirectory\Common7\IDE\ProjectTemplates\Programming Language\Locale ID

For example, the following directory contains the Visual Basic item templates for English (LCID 1033):

C:\VisualStudio\InstallationDirectory\Common7\IDE\ItemTemplates\VisualBasic\1033\

Organize templates

The categories in the **New Project** and **Add New Item** dialog boxes reflect the directory structures that exist in the installed template and user template locations. User templates can be organized into their own categories by adding new folders to the user template directory. The **New Project** and **Add New Item** dialog boxes reflect any changes you make to your user template categories.

NOTE

You cannot create a new category at the programming language level. New categories can only be created within each language.

To create new user project template categories

1. Create a folder in the programming language folder in the user project template directory. For example, to establish a **HelloWorld** category for C# project templates, create the following directory:

```
%USERPROFILE%\Documents\Visual Studio <Version>\Templates\ProjectTemplates\Visual  
C#\HelloWorld\
```

2. Place all the templates for this category in the new folder.

3. On the **File** menu, choose **New > Project**.

The **HelloWorld** category appears in the **New Project** dialog box, under **Installed > Visual C#**.

To create new user item template categories

1. Create a folder in the programming language folder in the user item template directory. For example, to establish a **HelloWorld** category for C# item templates, create the following directory:

```
%USERPROFILE%\Documents\Visual Studio <Version>\Templates\ItemTemplates\Visual C#\HelloWorld\
```

2. Place all the templates for this category in the new folder.

3. Create a project or open an existing project. Then, on the **Project** menu, choose **Add New Item**.

The **HelloWorld** category appears in the **Add New Item** dialog box, under **Installed > Visual C# Items**.

Display templates in parent categories

You can enable templates in subcategories to be displayed in their parent categories by using the

`NumberOfParentCategoriesToRollUp` element in the .vstemplate file. These steps are identical for project templates and item templates.

To display templates in parent categories

1. Locate the .zip file that contains the template.

2. Extract the .zip file.

3. Open the .vstemplate file in Visual Studio.

4. In the `TemplateData` element, add a `NumberOfParentCategoriesToRollUp` element. For example, the following code makes the template visible in the parent category, but no higher.

```
<TemplateData>  
  ...  
  <NumberOfParentCategoriesToRollUp>  
    1  
  </NumberOfParentCategoriesToRollUp>  
  ...  
</TemplateData>
```

5. Save and close the .vstemplate file.

6. Select the files in your template, right-click the selection, and choose **Send to > Compressed (zipped) folder**.

The files are compressed into a .zip file.

7. Delete the extracted template files and the old template .zip file.

8. Put the new .zip file in the directory that had the deleted .zip file.

See also

[Customizing Templates](#)

[Visual Studio Template Schema Reference \(Extensibility\)](#)

[NumberOfParentCategoriesToRollUp \(Visual Studio Templates\)](#)

[How to: Create Project Templates](#)

[How to: Create Item Templates](#)

Customizing project and item templates

1/5/2018 • 1 min to read • [Edit Online](#)

Even after project and item templates have been created, you can further customize them to meet your needs.

For example, you can perform the following tasks:

- Modify and export an existing template as a user template.

For more information, see [How to: Update Existing Templates](#).

- Pass custom parameters into a template to replace existing values.

For more information, see [How to: Substitute Parameters in a Template](#).

- Customize the wizards that create projects from templates.

For more information, see [How to: Use Wizards with Project Templates \(Extensibility\)](#).

Related sections

[Creating Project and Item Templates](#)

[How to: Troubleshoot Templates](#)

[How to: Create Project Templates](#)

[How to: Create Item Templates](#)

[How to: Create Starter Kits](#)

[Visual Studio Template Schema Reference](#)

[IWizard](#)

How to: Update existing templates

1/5/2018 • 1 min to read • [Edit Online](#)

After you create a template and compress the files into a .zip file, you may want to modify the template. You can do this by manually changing the files in the template, or by exporting a new template from a project that is based on the template.

Using the Export Template Wizard to update an existing project template

Visual Studio provides an **Export Template Wizard** that can be used to update an existing template:

1. Open the **New Project** dialog box by choosing **File > New > Project**.
2. Select the template that you want to update, enter a name and location for your project, and choose **OK**.
3. Modify the project in Visual Studio.
4. On the **Project** menu, choose **Export Template....**

The **Export Template Wizard** opens.
5. Follow the prompts in the wizard to export the template as a .zip file.
6. (Optional) To add the template to the **New Project** dialog box, place the .zip file in the following directory:
%USERPROFILE%\Documents\Visual Studio <version>\Templates\ProjectTemplates. You'll need to perform this step if you did not select the option **Automatically import the template into Visual Studio** in the **Export Template Wizard**.
7. Delete the old template .zip file.

Manually updating an existing template

You can update an existing template without using the **Export Template Wizard**, by modifying the files in the compressed .zip file.

To manually update an existing template

1. Locate the .zip file that contains the template. User project templates are usually located at
%USERPROFILE%\Documents\Visual Studio <version>\Templates\ProjectTemplates.
2. Extract the .zip file.
3. Modify or delete the current template files, or add new files to the template.
4. Open, modify, and save the .vstemplate XML file to handle updated behavior or new files.

For more information about the .vstemplate schema, see [Visual Studio template schema reference \(Extensibility\)](#). For more information about what you can parameterize in the source files, see [Template parameters](#).

5. Select the files in your template, and from the right-click or context menu, and choose **Send to > Compressed (zipped) folder**.

The files that you selected are compressed into a .zip file.

6. Put the new .zip file in the same directory as the old .zip file.
7. Delete the extracted template files and the old template .zip file.

See also

- [Customizing Templates](#)
- [Creating Project and Item Templates](#)
- [Visual Studio Template Schema Reference](#)
- [Template Parameters](#)
- [How to: Create Starter Kits](#)

How to: Substitute parameters in a template

1/5/2018 • 1 min to read • [Edit Online](#)

Template parameters enable you to replace identifiers such as class names and namespaces, when a file is created from a template. You can add template parameters to existing templates, or create your own templates with template parameters.

Template parameters are written in the format `$parameter$`. For a complete list of template parameters, see [Template parameters](#).

The following section shows you how to modify a template to replace the name of a namespace with the "safe project name".

To use a parameter to replace the namespace name

1. Insert the parameter in one or more of the code files in the template. For example:

```
namespace $safeprojectname$
```

2. In the .vstemplate file for the template, locate the `ProjectItem` element that includes this file.

3. Set the `ReplaceParameters` attribute to `true` for the `ProjectItem` element:

```
<ProjectItem ReplaceParameters="true">Class1.cs</ProjectItem>
```

See also

[Creating Project and Item Templates](#)

[Template Parameters](#)

[Visual Studio Template Schema Reference](#)

[ProjectItem Element \(Visual Studio Item Templates\)](#)

Visual Studio IDE 64-Bit Support

2/28/2018 • 1 min to read • [Edit Online](#)

Visual Studio enables you to set up your applications to target different platforms, including 64-bit platforms. For more information on 64-bit platform support in Visual Studio, see [64-bit Applications](#).

Deploying a 64-bit application

[Deploying Prerequisites for 64-bit Applications](#) lists the redistributables you can use as prerequisites for the installation of a 64-bit application.

Configuring projects as 64-bit applications

[How to: Configure Projects to Target Platforms](#) discusses configuring projects to be built as 64-bit applications.

Debugging a 64-bit application

[Debug 64-Bit Applications Using Dump Files](#)

Develop code in Visual Studio without projects or solutions

2/23/2018 • 4 min to read • [Edit Online](#)

In Visual Studio 2017, you can open code from nearly any type of directory-based project into Visual Studio without the need for a solution or project file. This means you can, for example, clone a repo on GitHub, open it directly into Visual Studio, and begin developing, without having to create a solution or project. If needed, you can specify custom build tasks and launch parameters through simple JSON files.

After you open your code files in Visual Studio, Solution Explorer displays all the files in the folder. You can click on any file to begin editing it. In the background, Visual Studio starts indexing the files to enable IntelliSense, navigation, and refactoring features. As you edit, create, move, or delete files, Visual Studio tracks the changes automatically and continuously updates its IntelliSense index. Code will appear with syntax colorization and, in many cases, include basic IntelliSense statement completion.

Open any code

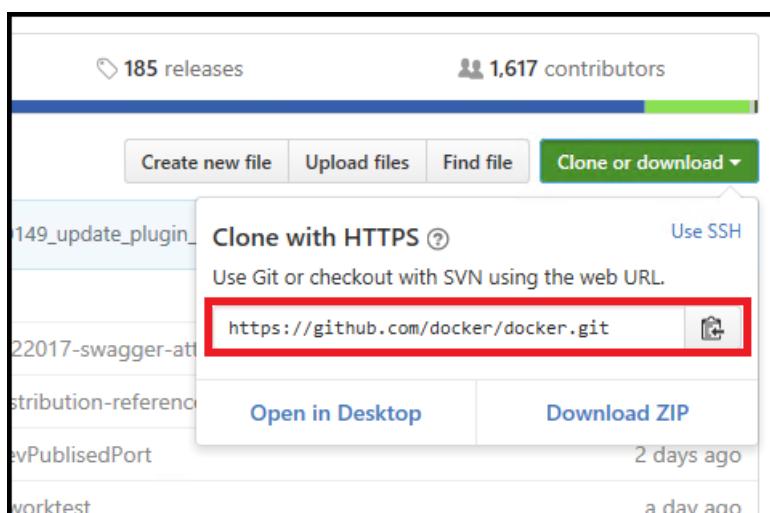
You can open code into Visual Studio in any of the following ways:

- On the Visual Studio menu bar, choose **File > Open > Folder**, and then browse to the code location.
- On the context (right-click) menu of a folder containing code, choose the **Open in Visual Studio** command.
- Choose the **Open Folder** link on the Visual Studio Start Page.
- If you are a keyboard user, press **Ctrl+Shift+Alt+O** in Visual Studio.
- Open code from a cloned GitHub repo.

To open code from a cloned GitHub repo

The following example shows how to clone a GitHub repo and then open its code in Visual Studio. To follow this procedure, you must have a GitHub account and Git for Windows installed on your system. See [Signing up for a new GitHub account](#) and [Git for Windows](#) for more information.

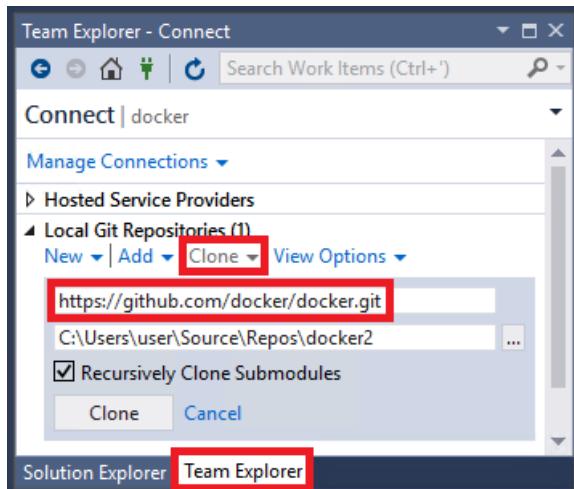
1. Go to the repo you want to clone on GitHub.
2. Choose the **Clone or Download** button and then choose the **Copy to Clipboard** button in the dropdown menu to copy the secure URL for the GitHub repo.



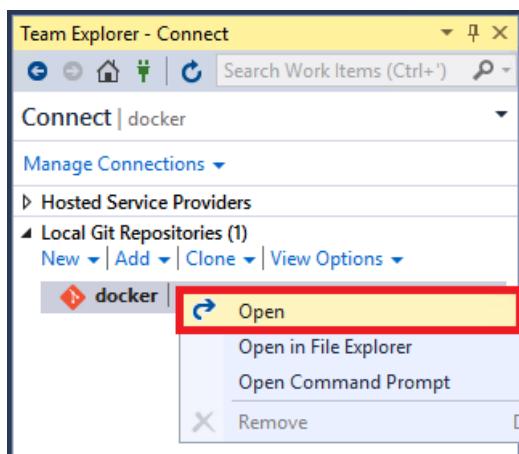
3. In Visual Studio, choose the **Team Explorer** tab to open Team Explorer. If you don't see the tab, open it

from **View > Team Explorer**.

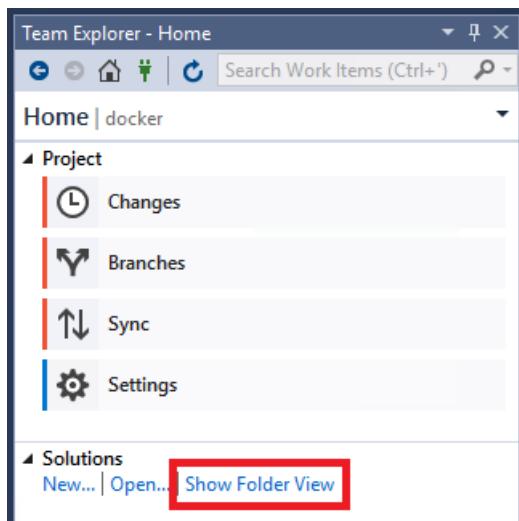
4. In Team Explorer, under the **Local Git Repositories** section, choose the **Clone** command and then paste the URL of the GitHub page into the text box.



5. Choose the **Clone** button to clone the project's files to a local Git repository. Depending on the size of the repo, this process could take several minutes.
6. After the repo has been cloned to your system, in Team Explorer, choose the **Open** command on the context (right-click) menu of the newly cloned repo.



7. Choose the **Show Folder View** command to view the files in Solution Explorer



You can now browse folders and files in the cloned repo, and view and search the code in the Visual Studio code editor, complete with syntax colorization and other features.



[Watch a video](#) on how to clone and open code from a GitHub repo in Visual Studio.

Run and debug your code

You can debug your code in Visual Studio without a project or solution! To debug some languages, you may need to specify a valid *startup file* in the codebase, such as a script, executable, or project. The drop-down list box next to the **Start** button on the toolbar lists all of the startup items that Visual Studio detects, as well as items you specifically designate. Visual Studio runs this code first when you debug your code.

Configuring your code to run in Visual Studio differs depending on what kind of code it is, and what the build tools are.

Codebases that use MSBuild

MSBuild-based codebases can have multiple build configurations that appear in the **Start** button's drop-down list. Select the file that you want to use as the startup item, and then choose the **Start** button to begin debugging.

NOTE

For C# and Visual Basic codebases, you must have the **.NET desktop development** workload installed. For C++ codebases, you must have the **Desktop development with C++** workload installed.

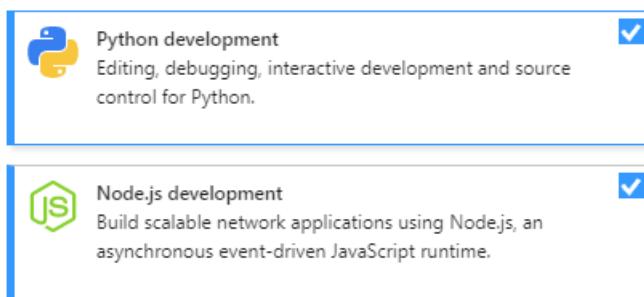
Codebases that use custom build tools

If your codebase uses custom build tools, then you must tell Visual Studio how to build your code using *build tasks* that are defined in a *json* file. For more information, see [Customize build and debug tasks](#).

Codebases that contain Python or JavaScript code

If your codebase contains Python or JavaScript code, you don't have to configure any *json* files, but you do have to install the corresponding workload. You must also configure the startup script:

1. Install the [Node.js development](#) or [Python development](#) workload by choosing **Tools > Get Tools and Features...**, or by closing Visual Studio and running the Visual Studio Installer.



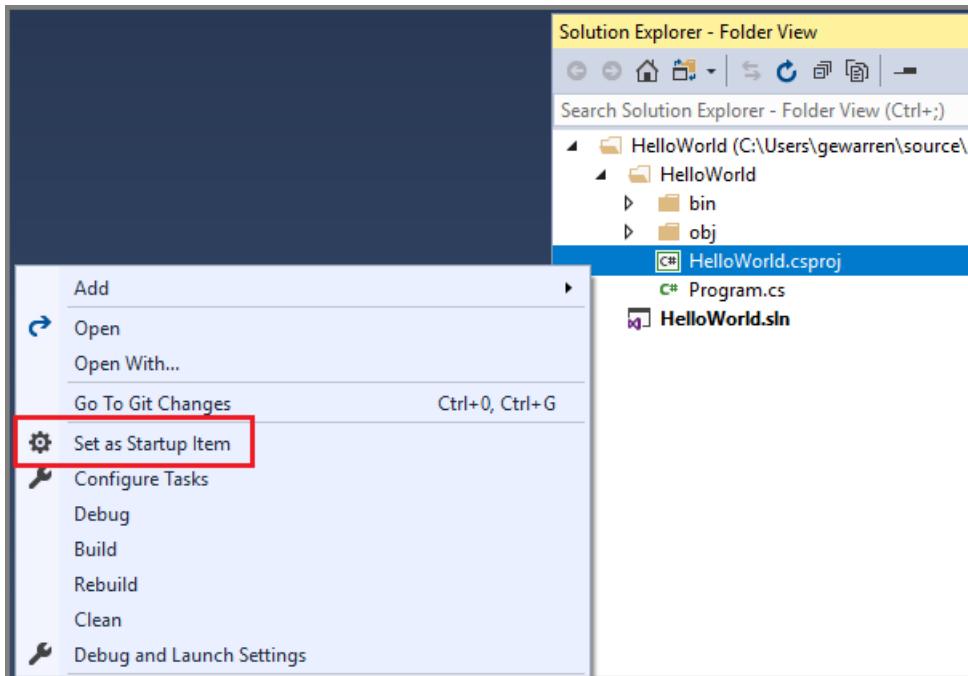
2. In **Solution Explorer**, on the right-click or context menu of a JavaScript or Python file, choose the **Set as Startup Item** command.
3. Choose the **Start** button to begin debugging.

Codebases that contain C++ code

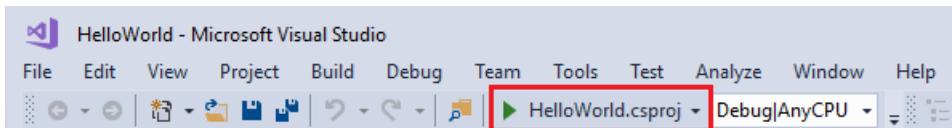
For information about opening C++ code without solutions or projects in Visual Studio, see [Open Folder projects for C++](#).

Codebases that contain a Visual Studio project

If your code folder contains a Visual Studio project, you can designate the project as the startup item.



The **Start** button's text changes to reflect that the project is the startup item.



See also

- [Customize build and debug tasks](#)
- [Open Folder projects for C++](#)
- [CMake projects in C++](#)
- [Writing code in the code and text editor](#)

Customize build and debug tasks for "Open Folder" development

2/23/2018 • 10 min to read • [Edit Online](#)

Visual Studio knows how to run many different languages and codebases, but it doesn't know how to run everything. If you [opened a code folder](#) in Visual Studio, and Visual Studio knows how to run your code, you can run it right away without any additional configuration.

If the codebase uses custom build tools that Visual Studio doesn't recognize, you need to provide some configuration details to run and debug the code in Visual Studio. You instruct Visual Studio how to build your code by defining *build tasks*. You can create one or more build tasks to specify all the items a language needs to build and run its code. You can also create arbitrary tasks that can do nearly anything you want. For example, you can create a task to list the contents of a folder or to rename a file.

Customize your project-less codebase by using the following *json* files:

| FILE NAME | PURPOSE |
|---------------------------------|---|
| <i>tasks.vs.json</i> | Specify custom build commands and compiler switches, and arbitrary (non-build related) tasks. Accessed via the Solution Explorer context menu item Configure Tasks . |
| <i>launch.vs.json</i> | Specify command-line arguments for debugging. Accessed via the Solution Explorer context menu item Debug and Launch Settings . |
| <i>VSWorkspaceSettings.json</i> | Generic settings that may impact tasks and launch. For example, defining <code>envVars</code> in <i>VSWorkspaceSettings.json</i> adds the specified environment variables to externally run commands. You create this file manually. |

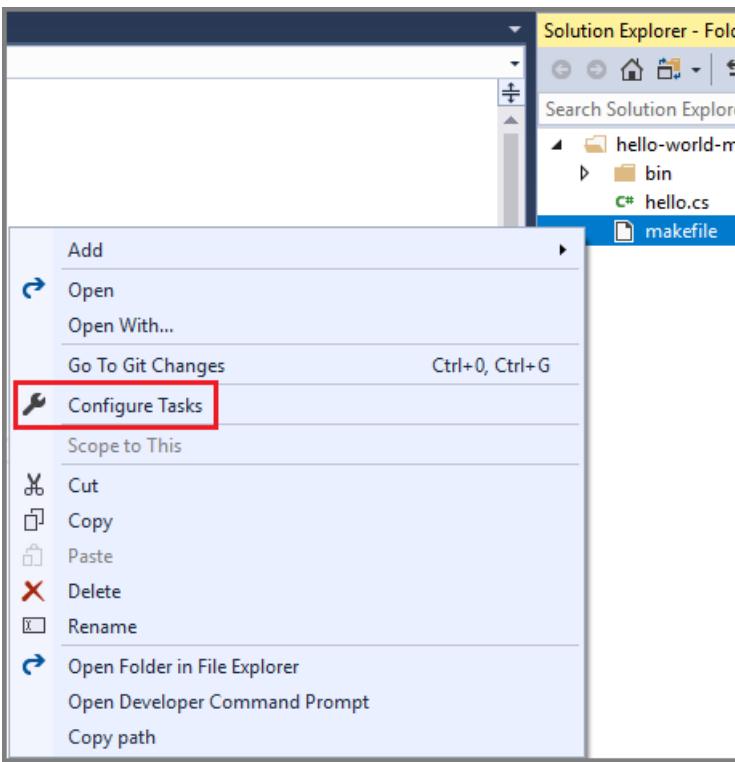
These *json* files are located in a hidden folder called `.vs` in the root folder of your codebase. The *tasks.vs.json* and *launch.vs.json* files are created by Visual Studio on an as-needed basis when you choose either **Configure Tasks** or **Debug and Launch Settings** on a file or folder in **Solution Explorer**. These *json* files are hidden because users generally don't want to check them into source control. However, if you want to be able to check them into source control, drag the files into the root of your codebase, where they are visible.

TIP

To view hidden files in Visual Studio, choose the **Show All Files** button on the Solution Explorer toolbar.

Define tasks with *tasks.vs.json*

You can automate build scripts or any other external operations on the files you have in your current workspace by running them as tasks directly in the IDE. You can configure a new task by right-clicking on a file or folder and selecting **Configure Tasks**.



This creates (or opens) the `tasks.vs.json` file in the `.vs` folder. You can define a build task or arbitrary task in this file, and then invoke it using the name you gave it from the **Solution Explorer** context menu.

Custom tasks can be added to individual files, or to all files of a specific type. For instance, NuGet package files can be configured to have a "Restore Packages" task, or all source files can be configured to have a static analysis task, such as a linter for all `.js` files.

Define custom build tasks

If your codebase uses custom build tools that Visual Studio doesn't recognize, then you cannot run and debug the code in Visual Studio until you complete some configuration steps. Visual Studio provides *build tasks* where you can tell Visual Studio how to build, rebuild, and clean your code. The `tasks.vs.json` build task file couples the Visual Studio inner development loop to the custom build tools used by your codebase.

Consider a codebase that consists of a single C# file called `hello.cs`. The makefile for such a codebase might look like this:

```
build: directory hello.exe

hello.exe: hello.cs
    csc -debug hello.cs /out:bin\hello.exe

clean:
    del bin\hello.exe bin\hello.pdb

rebuild: clean build

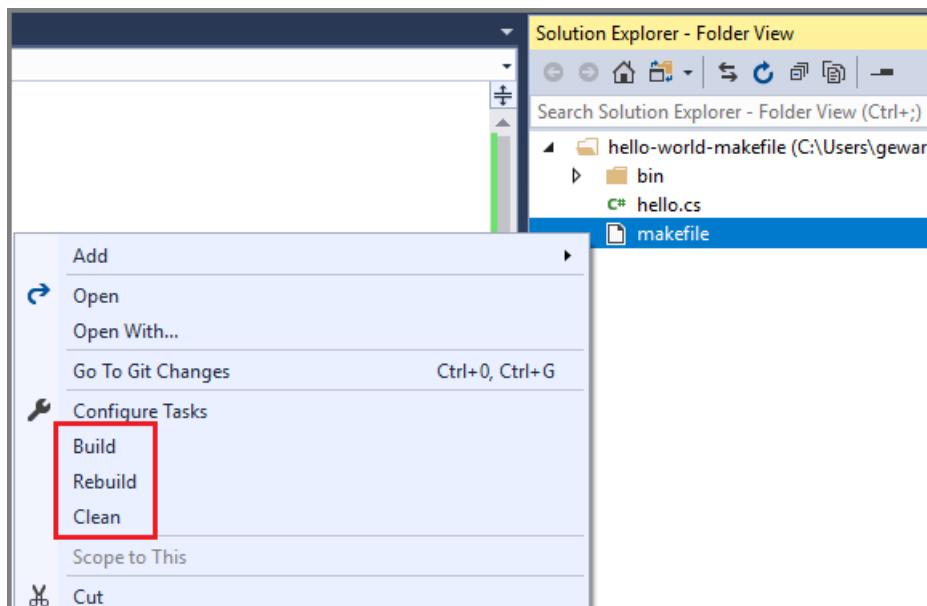
directory: bin

bin:
    md bin
```

For such a makefile that contains build, clean, and rebuild targets, you can define the following `tasks.vs.json` file. It contains three build tasks for building, rebuilding, and cleaning the codebase, using NMAKE as the build tool.

```
{
  "version": "0.2.1",
  "outDir": "${workspaceRoot}\\bin\\",
  "tasks": [
    {
      "taskName": "makefile-build",
      "appliesTo": "makefile",
      "type": "launch",
      "contextType": "build",
      "command": "nmake",
      "args": [ "build" ],
      "envVars": {
        "VSCMD_START_DIR": "${workspaceRoot}\\"
      }
    },
    {
      "taskName": "makefile-clean",
      "appliesTo": "makefile",
      "type": "launch",
      "contextType": "clean",
      "command": "nmake",
      "args": [ "clean" ],
      "envVars": {
        "VSCMD_START_DIR": "${workspaceRoot}\\"
      }
    },
    {
      "taskName": "makefile-rebuild",
      "appliesTo": "makefile",
      "type": "launch",
      "contextType": "rebuild",
      "command": "nmake",
      "args": [ "rebuild" ],
      "envVars": {
        "VSCMD_START_DIR": "${workspaceRoot}\\"
      }
    }
  ]
}
```

After you define build tasks in `tasks.json`, additional context menu items are added to the corresponding files in **Solution Explorer**. For this example, **Build**, **Rebuild**, and **Clean** options are added to the context menu of any `makefile` files.



NOTE

The commands appear in the context menu under the **Configure Tasks** command due to their `contextType` settings. "build", "rebuild", and "clean" are build commands, so they appear in the build section in the middle of the context menu.

When you select one of these options, the task executes. Output appears in the **Output** window, and build errors appear in the **Error List**.

Define arbitrary tasks

You can define arbitrary tasks in the `tasks.json` file, to do just about anything you want. For example, you can define a task to display the name of the currently selected file in the **Output** window, or to list the files in a specified directory.

The following example shows a `tasks.json` file that defines a single task. When invoked, the task displays the filename of the currently selected `js` file.

```
{
  "version": "0.2.1",
  "tasks": [
    {
      "taskName": "Echo filename",
      "appliesTo": "*.*",
      "type": "default",
      "command": "${env.COMSPEC}",
      "args": [ "echo ${file}" ]
    }
  ]
}
```

- `taskName` specifies the name that appears in the context menu.
- `appliesTo` specifies which files the command can be performed on.
- The `command` property specifies the command to invoke. In this example, the `COMSPEC` environment variable is used to identify the command line interpreter, typically `cmd.exe`.
- The `args` property specifies the arguments to be passed to the invoked command.
- The `${file}` macro retrieves the selected file in **Solution Explorer**.

After saving `tasks.json`, you can right-click on any `js` file in the folder and choose **Echo filename**. The file name is displayed in the **Output** window.

NOTE

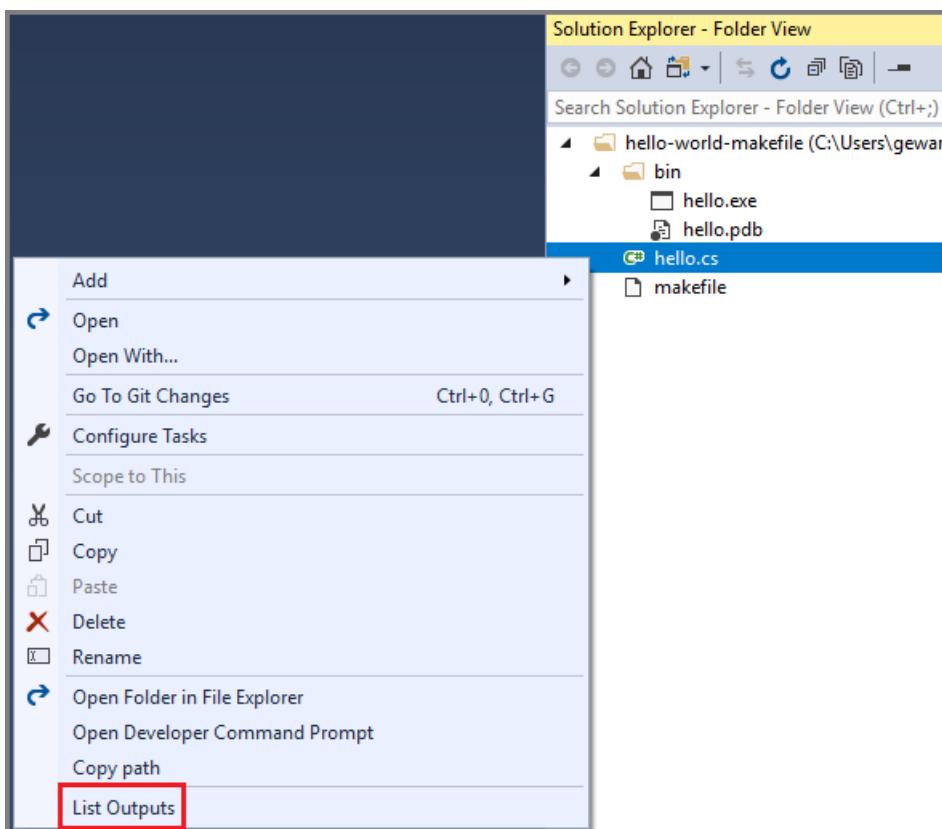
If your codebase doesn't contain a `tasks.json` file, you can create one by choosing **Configure Tasks** from the right-click or context menu of a file in **Solution Explorer**.

The next example defines a task that lists the files and subfolders of the `bin` directory.

```
{
  "version": "0.2.1",
  "outDir": "${workspaceRoot}\\bin\\",
  "tasks": [
    {
      "taskName": "List Outputs",
      "appliesTo": "*",
      "type": "default",
      "command": "${env.COMSPEC}",
      "args": [ "dir ${outDir}" ]
    }
  ]
}
```

- `${outDir}` is a custom macro that is first defined before the `tasks` block. It is then called in the `args` property.

This task applies to all files. When you open the context menu on any file in **Solution Explorer**, the task's name **List Outputs** appears at the bottom of the menu. When you choose **List Outputs**, the contents of the *bin* directory are listed in the **Output** window in Visual Studio.



Settings scope

Multiple `tasks.json` files can exist at the root and subdirectories of a codebase. This design enables the flexibility to have different behavior in different subdirectories of the codebase. Visual Studio aggregates or overrides settings throughout the codebase, prioritizing files in the following order:

- Settings files in the root folder's `.vs` directory.
- The directory where a setting is being computed.
- The current directory's parent directory, all the way up to the root directory.
- Settings files in the root directory.

These aggregation rules apply to `tasks.json` and `VSWorkspaceSettings.json` files. For information on how settings in other file are aggregated, see the corresponding section for that file in this article.

Properties for `tasks.json`

This section describes some of the properties you can specify in `tasks.json`.

appliesTo

You can create tasks for any file or folder by specifying its name in the `appliesTo` field, for example

`"appliesTo": "hello.js"`. The following file masks can be used as values:

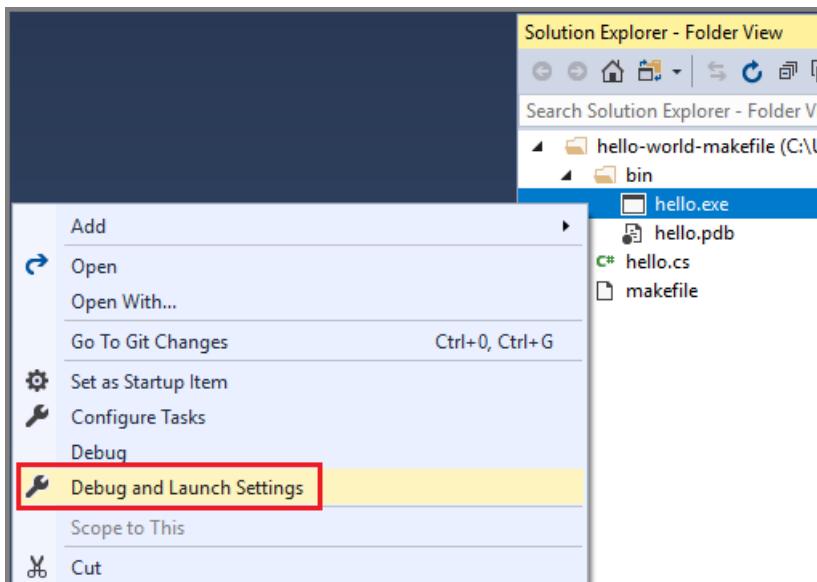
| | |
|--------------------------|---|
| <code>"*"</code> | task is available to all files and folders in the workspace |
| <code>"*/"</code> | task is available to all folders in the workspace |
| <code>".js"</code> | task is available to all files with the extension <code>.js</code> in the workspace |
| <code>"/*.js"</code> | task is available to all files with the extension <code>.js</code> in the root of the workspace |
| <code>"src/*/"</code> | task is available to all subfolders of the "src" folder |
| <code>"makefile"</code> | task is available to all makefile files in the workspace |
| <code>"/makefile"</code> | task is available only to the makefile in the root of the workspace |

Macros for `tasks.json`

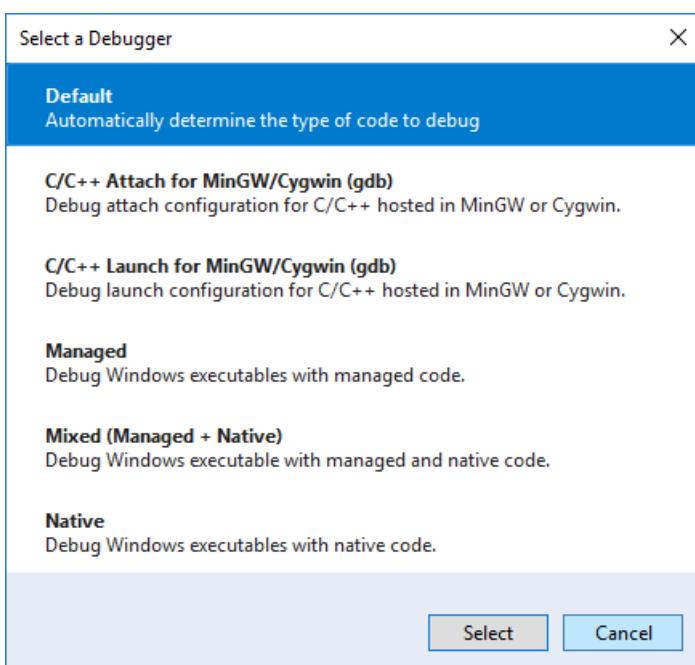
| | |
|---|---|
| <code> \${env.<VARIABLE>} </code> | Specifies any environment variable (for example, <code> \${env.PATH}</code> , <code> \${env.COMSPEC}</code> and so on) that is set for the developer command prompt. For more information, see Developer Command Prompt for Visual Studio . |
| <code> \${workspaceRoot} </code> | The full path to the workspace folder (for example, <code> "C:\sources\hello"</code>) |
| <code> \${file} </code> | The full path of the file or folder selected to run this task against (for example, <code> "C:\sources\hello\src\hello.js"</code>) |
| <code> \${relativeFile} </code> | The relative path to the file or folder (for example, <code> "src\hello.js"</code>) |
| <code> \${fileBasename} </code> | The name of the file without path or extension (for example, <code> "hello"</code>) |
| <code> \${fileDirname} </code> | The full path to the file, excluding the filename (for example, <code> "C:\sources\hello\src"</code>) |
| <code> \${fileExtname} </code> | The extension of the selected file (for example, <code> ".js"</code>) |

Configure debugging with `launch.json`

- To configure your codebase for debugging, in **Solution Explorer** choose the **Debug and Launch Settings** menu item from the right-click or context menu of your executable file.



2. In the **Select a Debugger** dialog box, choose an option, and then choose the **Select** button.

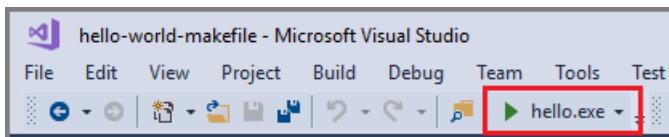


If the *launch.json* file doesn't already exist, it is created.

```
{  
    "version": "0.2.1",  
    "defaults": {},  
    "configurations": [  
        {  
            "type": "default",  
            "project": "bin\\hello.exe",  
            "name": "hello.exe"  
        }  
    ]  
}
```

3. Next, right-click on the executable file in **Solution Explorer**, and choose **Set as Startup Item**.

The executable is designated as the startup item for your codebase, and the debugging **Start** button's title changes to reflect the name of your executable.



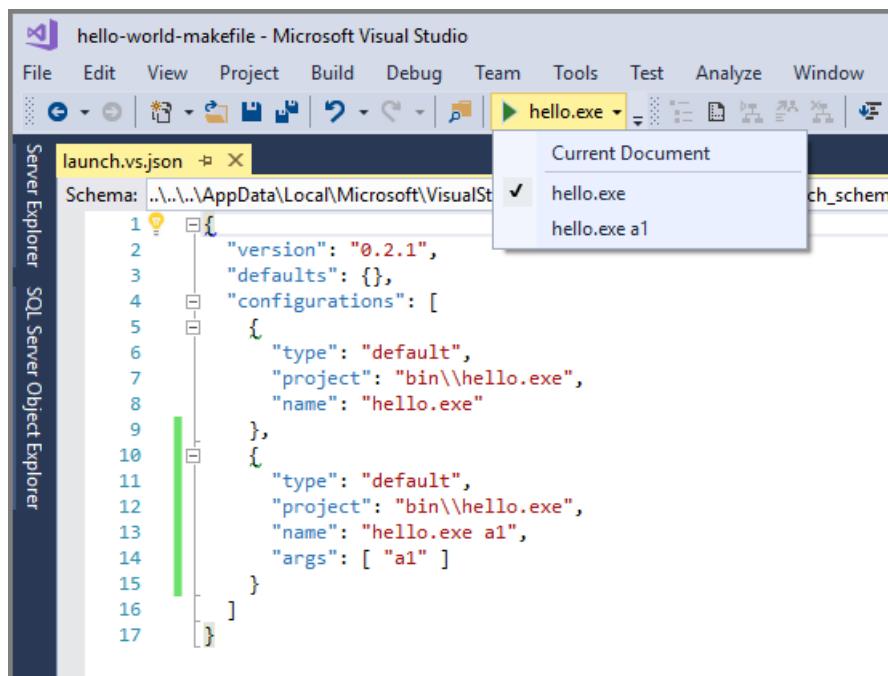
When you choose **F5**, the debugger launches and stops at any breakpoint you may have already set. All the familiar debugger windows are available and functional.

Specify arguments for debugging

You can specify command-line arguments to pass in for debugging in the *launch.json* file. Add the arguments in the `args` array, as shown in the following example:

```
{  
  "version": "0.2.1",  
  "defaults": {},  
  "configurations": [  
    {  
      "type": "default",  
      "project": "bin\\hello.exe",  
      "name": "hello.exe"  
    },  
    {  
      "type": "default",  
      "project": "bin\\hello.exe",  
      "name": "hello.exe a1",  
      "args": [ "a1" ]  
    }  
  ]  
}
```

When you save this file, the name of the new configuration appears in the debug target drop-down list, and you can select it to start the debugger. You can create as many debug configurations as you like.



NOTE

The `configurations` array property in *launch.json* is read from two file locations—the root directory for the codebase, and the `.vs` directory. If there is a conflict, priority is given to the value in `.vs\launch.json`.

Define workspace settings in VSWorkspaceSettings.json

You can specify generic settings that may impact tasks and launch in the `VSWorkspaceSettings.json` file. For example, if you define `envVars` in `VSWorkspaceSettings.json`, Visual Studio adds the specified environment variables to commands that are run externally. To use this file, you must create it manually.

Additional settings files

In addition to the three `.json` files described in this topic, Visual Studio also reads settings from some additional files, if they exist in your codebase.

`.vscode\settings.json`

Visual Studio reads limited settings from a file named `settings.json`, if it is in a directory named `.vscode`. This functionality is provided for codebases that have previously been developed in Visual Studio Code. Currently, the only setting that is read from `.vscode\settings.json` is `files.exclude`, which filters files visually in Solution Explorer and from some search tools.

You can have any number of `.vscode\settings.json` files in your codebase. Settings read from this file are applied to the parent directory of `.vscode` and all of its subdirectories.

`.gitignore`

`.gitignore` files are used to tell Git which files to ignore; that is, which files and directories you don't want to check in. `.gitignore` files are usually included as part of a codebase so that the settings can be shared with all developers of the codebase. Visual Studio reads patterns in `.gitignore` files to filter items visually and from some search tools.

Settings read from the `.gitignore` file are applied to its parent directory and all subdirectories.

See also

- [Develop code without projects or solutions](#)
- [Open Folder projects for C++](#)
- [CMake projects in C++](#)
- [NMAKE reference](#)
- [Writing code in the code and text editor](#)

Features of the code editor

2/26/2018 • 9 min to read • [Edit Online](#)

The Visual Studio editor provides many features that make it easier for you to write and manage your code and text. You can expand and collapse different blocks of code by using outlining. You can learn more about the code by using IntelliSense, the **Object Browser**, and the Call Hierarchy. You can find code by using features such as **Go To**, **Go To Definition**, and **Find All References**. You can insert blocks of code with code snippets, and you can generate code by using features such as **Generate From Usage**. If you have never used the Visual Studio editor before, see [Editing Your Code](#) for a quick overview.

You can view your code in a number of different ways. By default, **Solution Explorer** shows your code organized by files. You can click on the **Class View** tab at the bottom of the window to view your code organized by classes.

You can search and replace text in single or multiple files. For more information, see [Finding and Replacing Text](#). You can use regular expressions to find and replace text. For more information, see [Using Regular Expressions in Visual Studio](#).

The different Visual Studio languages offer different sets of features, and in some cases the features behave differently in different languages. Many of these differences are specified in the descriptions of the features, but for more information you can see the sections on specific Visual Studio languages.

Editor features

| Syntax Coloring | <p>Some syntax elements of code and markup files are colored differently to distinguish them. For example, keywords (such as <code>using</code> in C# and <code>Imports</code> in Visual Basic) are one color, but types (such as <code>Console</code> and <code>Uri</code>) are another color. Other syntax elements are also colorized, such as string literals and comments. C++ uses color to differentiate among types, enumerations, and macros, among other tokens.</p> <p>You can see the default color for each type, and you can change the color for any specific syntax element in the Fonts and Colors, Environment, Options Dialog Box, which you can open from the Tools menu.</p> |
|-------------------------|--|
| Error and Warning Marks | <p>As you add code and build your solution, you may see (a) different-colored wavy underlines (known as squiggles) or (b) light bulbs appearing in your code. Red squiggles denote syntax errors, blue denotes compiler errors, green denotes warnings, and purple denotes other types of errors. Light bulbs suggest fixes for problems and make it easy to apply the fix.</p> <p>You can see the default color for each error and warning squiggle in the Tools > Options > Environment > Fonts and Colors dialog box. Look for Syntax Error, Compiler Error, Warning, and Other Error.</p> |

| | |
|-------------------------|---|
| Brace Matching | When the insertion point is placed on an open brace in a code file, both it and the closing brace are highlighted. This feature gives you immediate feedback on misplaced or missing braces. You can turn brace matching on or off with the Automatic Delimiter Highlighting setting (Tools > Options > Text Editor). You can change the highlight color in the FONT AND COLORS setting (Tools > Options > Environment). Look for Brace Matching (Highlight) or Brace Matching (Rectangle) . |
| Structure Visualizer | Dotted lines connect matching braces in code files, making it easier to see opening and closing brace pairs. This can help you find code in your codebase more quickly. You can turn these lines on or off with the Show structure guidelines in the Display section of the Tools > Options > Text Editor > General page. |
| Line Numbers | Line numbers can be displayed in the left margin of the code window. They are not displayed by default. You can turn this option on in the Text Editor All Languages settings (Tools > Options > Text Editor > All Languages). You can display line numbers for individual programming languages by changing the settings for those languages (Tools > Options > Text Editor > <language>). For line numbers to print, you must select Include line numbers in the Print dialog box. |
| Change Tracking | The color of the left margin allows you to keep track of the changes you have made in a file. Changes you have made since the file was opened but not saved are denoted by a yellow bar on the left margin (known as the selection margin). After you have saved the changes (but before closing the file), the bar turns green. If you undo a change after you have saved the file, the bar turns orange. To turn this feature off and on, change the Track changes option in the Text Editor settings (Tools > Options > Text Editor). |
| Selecting Code and Text | You can select text either in the standard continuous stream mode or in box mode, in which you select a rectangular portion of text instead of a set of lines. To make a selection in box mode, press ALT as you drag the mouse over the selection (or press ALT + SHIFT + <arrow key>). The selection includes all of the characters within the rectangle defined by the first character and the last character in the selection. Anything typed or pasted into the selected area is inserted at the same point on each line. |
| Zoom | You can zoom in or out in any code window by pressing and holding the CTRL key and moving the scroll wheel on the mouse (or CTRL + SHIFT + . to increase and CTRL + SHIFT + , to decrease). You can also use the Zoom box in the lower left corner of the code window to set a specific zoom percentage. The zoom feature does not work in tool windows. |

| | |
|----------------------|---|
| Virtual Space | By default, lines in Visual Studio editors end after the last character, so that the RIGHT ARROW key at the end of a line moves the cursor to the beginning of the next line. In some other editors a line does not end after the last character, and you can place your cursor anywhere on the line. You can enable virtual space in the editor in the Tools > Options > Text Editor > All Languages settings. Note that you can enable either Virtual Space or Word Wrap , but not both. |
| Printing | <p>You can use the options in the Print dialog box to include line numbers or hide collapsed regions of code when you print a file. In the Page Setup dialog box, you can also choose to print the full path and the name of the file by choosing Page header.</p> <p>You can set color printing options in the Tools > Options > Environment > Fonts and Colors dialog box. Choose Printer in the Show settings for list to customize color printing. You can specify different colors for printing a file than for editing a file.</p> |
| Global Undo and Redo | The Undo Last Global Action and Redo Last Global Action commands on the Edit menu undo or redo global actions that affect multiple files. Global actions include renaming a class or namespace, performing a find-and-replace operation across a solution, refactoring a database, or any other action that changes multiple files. You can apply the global undo and redo commands to actions in the current Visual Studio session, even after you close the solution in which an action was applied. |

Advanced editing features

You can find a number of advanced features on the **Edit > Advanced** menu on the toolbar. Not all of these features are available for all types of code files.

| | |
|-------------------------|--|
| Format Document | Sets the proper indentation of lines of code and moves curly braces to separate lines in the document. |
| Format Selection | Sets the proper indentation of lines of code and moves curly braces to separate lines in the selection. |
| Tabify Selected Lines | Changes leading spaces to tabs where appropriate. |
| Untabify Selected Lines | Changes leading tabs to spaces. If you want to convert all the spaces in your file to tabs (or all the tabs to spaces), you can use the <code>Edit.ConvertSpacesToTabs</code> and <code>Edit.ConvertTabsToSpaces</code> commands. These commands do not appear in Visual Studio menus, but you can call them from the Quick Access window or the command window. |
| Make Uppercase | Changes all characters in the selection to uppercase, or if there is no selection, changes the character at the insertion point to uppercase. |

| | |
|-------------------------------|--|
| Make Lowercase | Changes all characters in the selection to lowercase, or if there is no selection, changes the character at the insertion point to lowercase. |
| Move selected Lines Up | Moves the selected line up one line. Shortcut: ALT + UP ARROW. |
| Move Selected Lines Down | Moves the selected line down one line. Shortcut: ALT + DOWN ARROW. |
| Delete Horizontal White Space | Deletes tabs or spaces at the end of the current line. |
| View White Space | Displays spaces as raised dots, and tabs as arrows. The end of a file is displayed as a rectangular glyph. If Tools > Options > Text Editor > All Languages > Word Wrap > Show visible glyphs for word wrap is selected, that glyph is also displayed. |
| Word Wrap | Causes all the lines in a document to be visible in the code window. You can turn word wrap off and on in the Text Editor All Languages settings (Tools > Options > Text Editor > All Languages). |
| Comment Selection | Adds comment characters to the selection or the current line. |
| Uncomment Selection | Removes comment characters from the selection or the current line. |
| Increase Line Indent | Adds a tab (or the equivalent spaces) to the selected lines or the current line. |
| Decrease Line Indent | Removes a tab (or the equivalent spaces) from the selected lines or the current line. |
| Select Tag | In a document that contains tags (for example, XML or HTML), selects the tag. |
| Select Tag Content | In a document that contains tags (for example, XML or HTML), selects the content. |

Navigate and find code

You can move around in the code editor in several different ways, including navigating backwards and forwards to previous insertion points, viewing the definition of a type or member, and jumping to a specific method using the navigation bar. For more information see [Navigating code](#).

Finding references in your code base

To find where particular code elements are referenced throughout your codebase, you can use the **Find All References** command. Also, when you click on a type or member, the **reference highlighting** feature automatically highlights all references to that type or member. For more information, see [Finding references in your code](#).

Customize the editor

You can share your Visual Studio settings with another developer, have your settings conform to a standard, or return to Visual Studio default settings by using the **Import and Export Settings Wizard** command on the **Tools** menu. In the **Import and Export Settings Wizard**, you can change selected general settings or language and project-specific settings.

To define new hotkeys or redefine existing hotkeys, go to **Tools > Options > Environment > Keyboard**. For more information about hotkeys, see [Default keyboard shortcuts](#).

For more information about customizing the editor, see [Customizing the editor](#). For JavaScript-specific editor options, see [JavaScript editor options](#).

See also

- [Visual Studio IDE](#)
- [Get started with C++ in Visual Studio](#)
- [Get started with C# and ASP.NET in Visual Studio](#)
- [Get started with Python in Visual Studio](#)

Finding and Replacing Text

12/22/2017 • 3 min to read • [Edit Online](#)

You can find and replace text in the Visual Studio code editor, and certain text-based output windows such as the **Find Results** windows, using the **Find and Replace** control or **Find/Replace in Files**. You can also search and replace in some designer windows, such as the XAML designer and the Windows Forms designer, and tool windows.

You can scope searches to the current document, the current solution, or a custom set of folders. You can also specify a set of file name extensions for multi-file searches. You can customize search syntax by using .NET regular expressions.

To find and replace regular expressions, see [Using Regular Expressions in Visual Studio](#).

TIP

The **Find/Command** box is still available as a toolbar control, but is no longer visible by default. You can display the **Find/Command** box by choosing **Add or Remove Buttons** on the **Standard** toolbar and then choosing **Find**. For more information, see [Find/Command Box](#).

Find and Replace control

The **Find and Replace** control appears in the upper right corner of the code editor window. The **Find and Replace** control immediately highlights every occurrence of the given search string in the current document. You can navigate from one occurrence to another by choosing the **Find Next** button or the **Find Previous** button on the search control.

You can access replacement options by choosing the button next to the **Find** text box. To make one replacement at a time, choose the **Replace Next** button next to the **Replace** text box. To replace all matches, choose the **Replace All** button.

To change the highlight color for matches, choose the **Tools** menu, select **Options**, and then choose **Environment**, and select **Fonts and Colors**. In the **Show settings for** list, select **Text Editor**, and then in the **Display items** list, select **Find Highlight (Extension)**.

Searching Tool Windows

You can use the **Find** control in code or text windows, such as **Output** windows, and **Find Results** windows, by choosing **Find and Replace** on the **Edit** menu or (CTRL+F).

A version of the Find control is also available in some tool windows. For example, you can now filter the list of controls in the **Toolbox** window by entering text in the search box. Other tool windows that now allow you to search their contents include **Solution Explorer**, the **Properties** window, and **Team Explorer**, among others.

Find/Replace in Files

Find/Replace in Files works like the **Find and Replace** control, except that you can define a scope for your search. Not only can you search the current open file in the editor, but you can also search all open documents, the entire solution, the current project, and selected folder sets. You can also search by file name extension. To access the **Find/Replace in Files** dialog box, choose **Find and Replace** on the **Edit** menu (or CTRL+SHIFT+F).

When you choose **Find All**, a **Find Results** window opens and lists the matches for your search. Selecting a result

in the list displays the associated file and highlights the match. If the file is not already open for editing, it is opened in a preview tab in the right side of the tab well. You can use the **Find** control to search through the **Find Results** list.

Creating Custom Search Folder Sets

You can define a search scope by choosing the **Choose Search Folders** button (it looks like ...) next to the **Look in** box. In the **Choose Search Folders** dialog box, you can specify a set of folders in which to search, and you can save the specification so that you can reuse it later. You can specify folders on a remote machine only if you have mapped its drive to the local machine.

Creating Custom Component Sets

You can define component sets as your search scope by choosing the **Edit Custom Component Set** button next to the **Look in** box. You can specify installed .NET or COM components, Visual Studio projects that are included in your solution, or any assembly or type library (.dll, .tlb, .olb, .exe, or .ocx). To search references, select the **Look in references** box.

See Also

[Using Regular Expressions in Visual Studio](#)

Using regular expressions in Visual Studio

3/27/2018 • 3 min to read • [Edit Online](#)

Visual Studio uses [.NET Framework regular expressions](#) to find and replace text.

Replacement patterns

To use a numbered capture group, surround the group with parentheses in the regular expression pattern. Use `$number`, where `number` is an integer starting at 1, to specify a specific, numbered group in a replacement pattern. For example, the grouped regular expression `(\d)([a-z])` defines two groups: the first group contains a single decimal digit, and the second group contains a single character between **a** and **z**. The expression finds four matches in the following string: **1a 2b 3c 4d**. The replacement string `z$1` references the first group only, and converts the string to **z1 z2 z3 z4**.

For information about regular expressions that are used in replacement patterns, see [Substitutions in regular expressions \(.NET Guide\)](#).

Regular expression examples

Here are some examples:

| PURPOSE | EXPRESSION | EXAMPLE |
|---|------------|---|
| Match any single character (except a line break) | . | <code>a.o</code> matches "aro" in "around" and "abo" in "about" but not "acro" in "across". |
| Match zero or more occurrences of the preceding expression (match as many characters as possible) | * | <code>a*r</code> matches "r" in "rack", "ar" in "ark", and "aar" in "aardvark" |
| Match any character zero or more times (Wildcard *) | .* | <code>c.*e</code> matches "cke" in "racket", "comme" in "comment", and "code" in "code" |
| Match one or more occurrences of the preceding expression (match as many characters as possible) | + | <code>e.+e</code> matches "eede" in "feeder" but not "ee". |
| Match any character one or more times (Wildcard ?) | .+ | <code>e.+?e</code> matches "eede" in "feeder" but not "ee". |
| Match zero or more occurrences of the preceding expression (match as few characters as possible) | *? | <code>e.*?e</code> matches "ee" in "feeder" but not "eede". |
| Match one or more occurrences of the preceding expression (match as few characters as possible) | +? | <code>e.+?e</code> matches "ente" and "erprise" in "enterprise", but not the whole word "enterprise". |
| Anchor the match string to the beginning of a line or string | ^ | <code>^car</code> matches the word "car" only when it appears at the beginning of a line. |

| PURPOSE | EXPRESSION | EXAMPLE |
|---|--|--|
| Anchor the match string to the end of a line | \r?\$ | End\r?\$ matches "end" only when it appears at the end of a line. |
| Match any single character in a set | [abc] | b[abc] matches "ba", "bb", and "bc". |
| Match any character in a range of characters | [a-f] | be[n-t] matches "bet" in "between", "ben" in "beneath", and "bes" in "beside", but not "below". |
| Capture and implicitly number the expression contained within parenthesis | \1 | ([a-z])X\1 matches "aXa" and "bXb", but not "aXb". "\1" refers to the first expression group "[a-z]". |
| Invalidate a match | (?!abc) | real (?!ity) matches "real" in "realty" and "really" but not in "reality." It also finds the second "real" (but not the first "real") in "realtyreal". |
| Match any character that is not in a given set of characters | [^abc] | be[^n-t] matches "bef" in "before", "beh" in "behind", and "bel" in "below", but not "beneath". |
| Match either the expression before or the one after the symbol. | | (sponge|mud) bath matches "sponge bath" and "mud bath." |
| Escape the character following the backslash | \ | \^ matches the character ^. |
| Specify the number of occurrences of the preceding character or group | {x}, where x is the number of occurrences | x(ab){2}x matches "xababx", and x(ab){2,3}x matches "xababx" and "xabababx" but not "xababababx". |
| Match text in a Unicode character class, where "X" is the Unicode number. For more information about Unicode character classes, see Unicode Standard 5.2 Character Properties . | \p{X} | \p{Lu} matches "T" and "D" in "Thomas Doe". |
| Match a word boundary | \b (Outside a character class \b specifies a word boundary, and inside a character class specifies a backspace). | \bin matches "in" in "inside" but not "pinto". |
| Match a line break (that is, a carriage return followed by a new line). | \r?\n | End\r?\nBegin matches "End" and "Begin" only when "End" is the last string in a line and "Begin" is the first string in the next line. |
| Match any alphanumeric character | \w | a\wd matches "add" and "a1d" but not "a d". |
| Match any whitespace character. | (?([^\r\n])\s) | Public\sInterface matches the phrase "Public Interface". |

| PURPOSE | EXPRESSION | EXAMPLE |
|------------------------------|--|--|
| Match any numeric character | \d | \d matches and "3" in "3456", "2" in "23", and "1" in "1". |
| Match a Unicode character | \uXXXX where XXXX specifies the Unicode character value. | \u0065 matches the character "e". |
| Match an identifier | \b(\w+ [\w-[0-9]\]]\w*)\b | Matches "type1" but not &type1" or "#define". |
| Match a string inside quotes | (\".+?\") (''+?')) | Matches any string inside single or double quotes. |
| Match a hexadecimal number | \b0[xX]([0-9a-fA-F])\b | Matches "0xc67f" but not "0xc67fc67f". |
| Match integers and decimals | \b[0-9]*\.*[0-9]+\b | Matches "1.333". |

TIP

In Windows operating systems, most lines end in "\r\n" (a carriage return followed by a new line). These characters aren't visible, but are present in the editor and are passed to the .NET regular expression service.

See also

[Finding and Replacing Text](#)

Find/Command box

12/22/2017 • 1 min to read • [Edit Online](#)

You can search for text and run Visual Studio commands from the **Find/Command** box. The **Find/Command** box is still available as a toolbar control, but is no longer visible by default. You can display the **Find/Command** box by choosing **Add or Remove Buttons** on the **Standard** toolbar and then choosing **Find**.

To run a Visual Studio command, preface it with a greater than (>) sign.

The **Find/Command** box retains the last 20 items entered and displays them in a drop-down list. You can navigate through the list by choosing the arrow keys.



Searching for text

By default, when you specify text in the **Find/Command** box and then choose the **Enter** key, Visual Studio searches the current document or tool window using the options that are specified in the **Find in Files** dialog box. For more information, see [Finding and Replacing Text](#).

Entering commands

To use the **Find/Command** box to issue a single Visual Studio command or alias rather than search for text, preface the command with a greater than (>) symbol. For example:

```
>File.NewFile c:\temp\MyFile /t:"General\Text File"
```

Alternatively, you can also use the Command window to enter and execute single or multiple commands. Some commands or aliases can be entered and executed by themselves; others have required arguments in their syntax. For a list of commands that have arguments, see [Visual Studio Commands](#).

Escape Characters

A caret (^) character in a command means that the character immediately following it is interpreted literally, rather than as a control character. This can be used to embed straight quotation marks ("), spaces, leading slashes, carets, or any other literal characters in a parameter or switch value, with the exception of switch names. For example:

```
>Edit.Find ^^t /regex
```

A caret functions the same whether it is inside or outside quotation marks. If a caret is the last character on the line, it is ignored.

See also

[Command Window](#)

[Finding and Replacing Text](#)

Find in Files

12/22/2017 • 3 min to read • [Edit Online](#)

Find in Files allows you to search a specified set of files. The matches found and actions taken are listed in the **Find Results** window selected in **Result options**.

You can use any of the following methods to display **Find in Files** in the **Find and Replace** window.

To display Find in Files

1. On the menu bar, choose **Edit, Find and Replace**.
2. Choose **Find in Files**.

To cancel a Find operation, press **Ctrl + Break**.

NOTE

The Find and Replace tool does not search directories with the `Hidden` or `System` attribute.

Find what

To search for a new text string or expression, specify it in the box. To search for any of the 20 strings that you searched for most recently, open the drop-down list and choose the string. Choose the adjacent **Expression Builder** button if you want to use one or more regular expressions in your search string. For more information, see [Using Regular Expressions in Visual Studio](#).

NOTE

The **Expression Builder** button will only be enabled if you have selected **Use Regular Expressions** under **Find options**.

Look in

The option chosen from the **Look in** drop-down list determines whether **Find in Files** searches only in currently active files or in all files stored within certain folders. Select a search scope from the list or click the **Browse (...)** button to display the **Choose Search Folders** dialog box and to enter your own set of directories. You can also type a path directly into the **Look in** box.

WARNING

With the **Entire Solution** or **Current Project** options, project and solution files are not searched. If you want to look in project files, choose a search folder.

NOTE

If the **Look in** option selected causes you to search a file that you have checked out from source code control, only the version of that file which has been downloaded to your local machine is searched.

Include subfolders

Specifies that subfolders of the **Look in** folder will be searched.

Find options

You can expand or collapse the **Find options** section. The following options can be selected or cleared:

Match case

When selected, a **Find Results** search will be case-sensitive

Match whole word

When selected, the **Find Results** windows will only return whole word matches.

Use Regular Expressions

If this check box is selected, you can use special notations to define patterns of text to match in the **Find what** or **Replace with** text boxes. For a list of these notations, see [Using Regular Expressions in Visual Studio](#).

Look at these file types

This list indicates the types of files to search through in the **Look in** directories. If this field is blank, all of the files in the **Look in** directories will be searched.

Select any item in the list to enter a preconfigured search string that will find files of those particular types.

Result options

You can expand or collapse the **Result options** section. The following options can be selected or cleared:

Find results 1 window

When selected, the results of the current search will replace the content of the **Find Results 1** window. This window opens automatically to display your search results. To open this window manually, select **Other Windows** from the **View** menu and choose **Find Results 1**.

Find results 2 window

When selected, the results of the current search will replace the content of the **Find Results 2** window. This window opens automatically to display your search results. To open this window manually, select **Other Windows** from the **View** menu and choose **Find Results 2**.

Display file names only

Displays a list of files containing search matches rather than displaying the search matches themselves.

Append results

Appends the results from the search to the previous search results.

See also

[Finding and Replacing Text](#)

[Replace in Files](#)

[Visual Studio Commands](#)

Replace in Files

12/22/2017 • 4 min to read • [Edit Online](#)

Replace in Files allows you to search the code of a specified set of files for a string or expression, and change some or all of the matches found. The matches found and actions taken are listed in the **Find Results** window selected in **Result options**.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, for example to **General** or **Visual C++** settings, choose **Tools, Import and Export Settings**, and then choose **Reset all settings**.

You can use any of the following methods to display **Replace in Files** in the **Find and Replace** window.

To display Replace in Files

1. On the **Edit** menu, expand **Find and Replace**.
2. Choose **Replace in Files**.

— or —

If the **Find and Replace** window is already open, on the toolbar, choose **Replace in Files**.

Find what

To search for a new text string or expression, specify it in the box. To search for any of the 20 strings that you searched for most recently, open the drop-down list and choose the string. Choose the adjacent **Expression Builder** button if you want to use one or more regular expressions in your search string. For more information, see [Using Regular Expressions in Visual Studio](#).

NOTE

The **Expression Builder** button will only be enabled if you have selected **Use Regular Expressions** under **Find options**.

Replace With

To replace instances of the string in the **Find what** box with another string, enter the replacement string in the **Replace With** box. To delete instances of the string in the **Find what** box, leave this field blank. Open the list to display the 20 strings for which you searched most recently. Choose the adjacent **Expression Builder** button if you want to use one or more regular expressions in your replacement string. For more information, see [Using Regular Expressions in Visual Studio](#).

Look in

The option chosen from the **Look in** drop-down list determines whether **Replace in Files** searches only in currently active files or searches all files stored within certain folders. Select a search scope from the list, type a folder path, or click the **Browse (...)** button to display the **Choose Search Folders** dialog box and choose a set of folders to search. You can also type a path directly into the **Look in** box.

NOTE

If the **Look in** option selected causes you to search a file that you have checked out from source code control, only the version of that file which has been downloaded to your local machine is searched.

Find options

You can expand or collapse the **Find options** section. The following options can be selected or cleared:

Match case

When selected, the **Find Results** windows will only display instances of the **Find what** string that are matched both by content and by case. For example, a search for "MyObject" with **Match case** selected will return "MyObject" but not "myobject" or "MYOBJECT."

Match whole word

When selected, the **Find Results** windows will only display instances of the **Find what** string that are matched in complete words. For example, a search for "MyObject" will return "MyObject" but not "CMyObject" or "MyObjectC."

Use Regular Expressions

When this check box is selected, you can use special notations to define patterns of text in the **Find what** or **Replace with** text boxes. For a list of these notations, see [Using Regular Expressions in Visual Studio](#).

Look at these file types

This list indicates the types of files to search through in the **Look in** directories. If this field is left blank, all of the files in the **Look in** directories will be searched.

Select any item in the list to enter a preconfigured search string that will find files of those particular types.

Result options

You can expand or collapse the **Result options** section. The following options can be selected or cleared:

Find Results 1 window

When selected, the results of the current search will replace the content of the **Find Results 1** window. This window opens automatically to display your search results. To open this window manually, select **Other Windows** from the **View** menu and choose **Find Results 1**.

Find Results 2 window

When selected, the results of the current search will replace the content of the **Find Results 2** window. This window opens automatically to display your search results. To open this window manually, select **Other Windows** from the **View** menu and choose **Find Results 2**.

Display file names only

When this check box is selected, the Find Results windows list the full names and paths for all files that contain the search string. However, the results don't include the line of code where the string appears. This check box is available for Find in Files only.

Keep modified files open after Replace All

When selected, leaves open all files in which replacements have been made, so you can undo or save the changes. Memory constraints might limit the number of files that can remain open after a replace operation.

Caution

You can use **Undo** only on files that remain open for editing. If this option is not selected, files that were not already open for editing will remain closed, and no **Undo** option will be available in those files.

See also

[Finding and Replacing Text](#)

[Find in Files](#)

[Visual Studio Commands](#)

Encodings and Line Breaks

12/22/2017 • 1 min to read • [Edit Online](#)

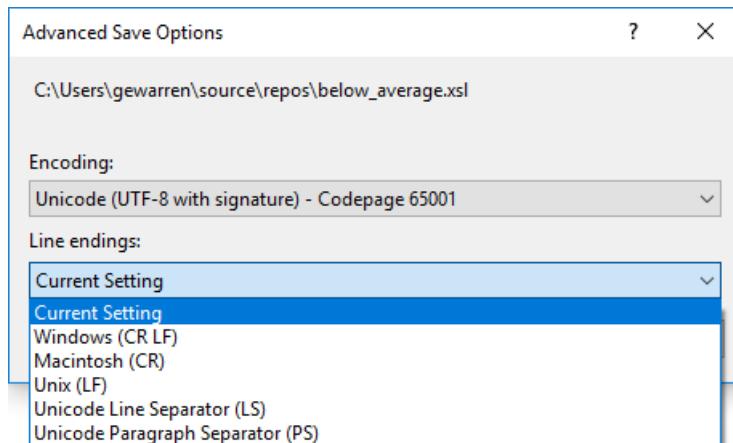
The following characters are interpreted as line breaks in Visual Studio:

- CR LF: Carriage return + line feed, Unicode characters 000D + 000A
- LF: Line feed, Unicode character 000A
- NEL: Next line, Unicode character 0085
- LS: Line separator, Unicode character 2028
- PS: Paragraph separator, Unicode character 2029

Text that is copied from other applications keeps the original encoding and line break characters. For example, when you copy text from Notepad and paste it into a text file in Visual Studio, the text has the same settings that it had in Notepad.

When you open a file that has different line break characters, you may see a dialog box that asks whether the inconsistent line break characters should be normalized, and which type of line breaks to choose.

You can use the **File, Advanced Save Options** dialog box to determine the type of line break characters you want. You can also change the encoding of a file with the same settings.



NOTE

If you don't see **Advanced Save Options** on the **File** menu, you can add it. Choose **Tools, Customize...**, and then choose the **Commands** tab. In the **Menu bar** drop-down list, choose **File**, then choose the **Add Command...** button. In the **Add Command** dialog box, under **Categories**, choose **File**, and then in the **Commands** list, choose **Advanced Save Options....**. Choose **OK** and then choose the **Move Down** button to move the command to any place in the menu. Choose **Close** to close the **Customize** dialog box. For more information, see [Customize menus and toolbars](#).

Alternatively, you can access the **Advanced Save Options** dialog box by choosing **File, Save <file> As....** In the **Save File As** dialog box, choose the drop-down triangle next to the **Save** button and choose **Save with encoding....**

See also

[Writing code in the editor](#)

How to: Save and Open Files with Encoding

12/22/2017 • 1 min to read • [Edit Online](#)

You can save files with specific character encoding to support bi-directional languages. You can also specify an encoding when opening a file, so that Visual Studio displays the file correctly.

To save a file with encoding

1. From the **File** menu, choose **Save File As**, and then click the drop-down button next to the **Save** button.

The **Advanced Save Options** dialog box is displayed.

2. Under **Encoding**, select the encoding to use for the file.
3. Optionally, under **Line endings**, select the format for end-of-line characters.

This option is useful if you intend to exchange the file with users of a different operating system.

If you want to work with a file that you know is encoded in a specific way, you can tell Visual Studio to use that encoding when opening the file. The method you use depends on whether the file is part of your project.

To open an encoded file that is part of a project

1. In **Solution Explorer**, right-click the file and choose **Open With**.
2. In the **Open With** dialog box, choose the editor to open the file with.

Many Visual Studio editors, such as the forms editor, will auto-detect the encoding and open the file appropriately. If you choose an editor that allows you to choose an encoding, the **Encoding** dialog box is displayed.

3. In the **Encoding** dialog box, select the encoding that the editor should use.

To open an encoded file that is not part of a project

1. On the **File** menu, point to **Open**, choose **File** or **File From Web**, and then select the file to open.
2. Click the drop-down button next to the **Open** button and choose **Open With**.
3. Follow Steps 2 and 3 from the preceding procedure.

See also

[Encoding and Line Breaks](#)

[Encoding and Windows Forms Globalization](#)

[Globalizing and Localizing Applications](#)

Outlining

12/22/2017 • 2 min to read • [Edit Online](#)

You can choose to hide some code from view by collapsing a region of code so that it appears under a plus sign (+). You expand a collapsed region by clicking the plus sign. If you are a keyboard user, you can choose **Ctrl + M + M** to collapse and expand. You can also collapse an outlining region by double-clicking any line in the region on the outlining margin, which appears just to the left of the code. You can see the contents of a collapsed region as a tooltip when you hover over the collapsed region.

Regions in the outlining margin are also highlighted when you hover over the margin with the mouse. The default highlighting color may seem rather faint in some color configurations. You can change it in **Tool, Options, Environment, Fonts and Colors, Collapsible Region**.

When you work in outlined code, you can expand the sections you want to work on, collapse them when you are done, and then move to other sections. When you do not wish to have outlining displayed, you can use the **Stop Outlining** command to remove the outline information without disturbing your underlying code.

The **Undo** and **Redo** commands on the **Edit** menu affect these actions. The **Copy**, **Cut**, **Paste**, and drag-and-drop operations retain outlining information, but not the state of the collapsible region. For example, when you copy a region that is collapsed, the **Paste** operation will paste the copied text as an expanded region.

Caution

When you change an outlined region, the outlining may be lost. For example, deletions or Find and Replace operations may erase the end of the region.

The following commands can be found on the **Edit, Outlining** submenu.

| | |
|----------------------------|--|
| Hide Selection | (CTRL + M , CTRL + H) - Collapses a selected block of code that would not normally be available for outlining, for example an <code>if</code> block. To remove the custom region, use Stop Hiding Current (or CTRL + M, CTRL + U). Not available in Visual Basic. |
| Toggle Outlining Expansion | - Reverses the current hidden or expanded state of the innermost outlining section when the cursor lies in a nested collapsed section. |
| Toggle All Outlining | (CTRL + M, CTRL + L) - Sets all regions to the same collapsed or expanded state. If some regions are expanded and some collapsed, then the collapsed regions are expanded. |
| Stop Outlining | (CTRL + M, CTRL + P) - Removes all outlining information for the entire document. |
| Stop Hiding Current | (CTRL + M, CTRL + U) - Removes the outlining information for the currently selected user-defined region. Not available in Visual Basic. |
| Collapse to Definitions | (CTRL + M, CTRL + O) - Collapses the members of all types. |

| | |
|-----------------------------------|---|
| Collapse Block:<logical boundary> | (Visual C++) Collapses a region in the function containing the insertion point. For example, if the insertion point lies inside a loop, the loop is hidden. |
|-----------------------------------|---|

Collapse All in: <logical structures>

(Visual C++) Collapses all the structures inside the function.

You can also use the Visual Studio SDK to define the text regions you want to expand or collapse. See

[Walkthrough: Outlining](#).

See also

[Writing code in the editor](#)

Code generation features in Visual Studio

2/6/2018 • 1 min to read • [Edit Online](#)

There are numerous ways that Visual Studio can help you generate, fix, and refactor code.

- You can use [code snippets](#) to insert a template such as a [switch](#) block or an [enum](#) declaration.
- You can use [Quick Actions](#) to generate code such as classes and properties, or to introduce a local variable. You can also use Quick Actions to [improve code](#), for example to remove unnecessary casts and unused variables, or to add null checks before accessing variables.
- You can [refactor code](#) to rename a variable, re-order method parameters, or synchronize a type with its filename, to name a few.

NOTE

Each language service in Visual Studio provides its own code generation capabilities, so some features are only available in C#, and some are available in both C# and Visual Basic.

See also

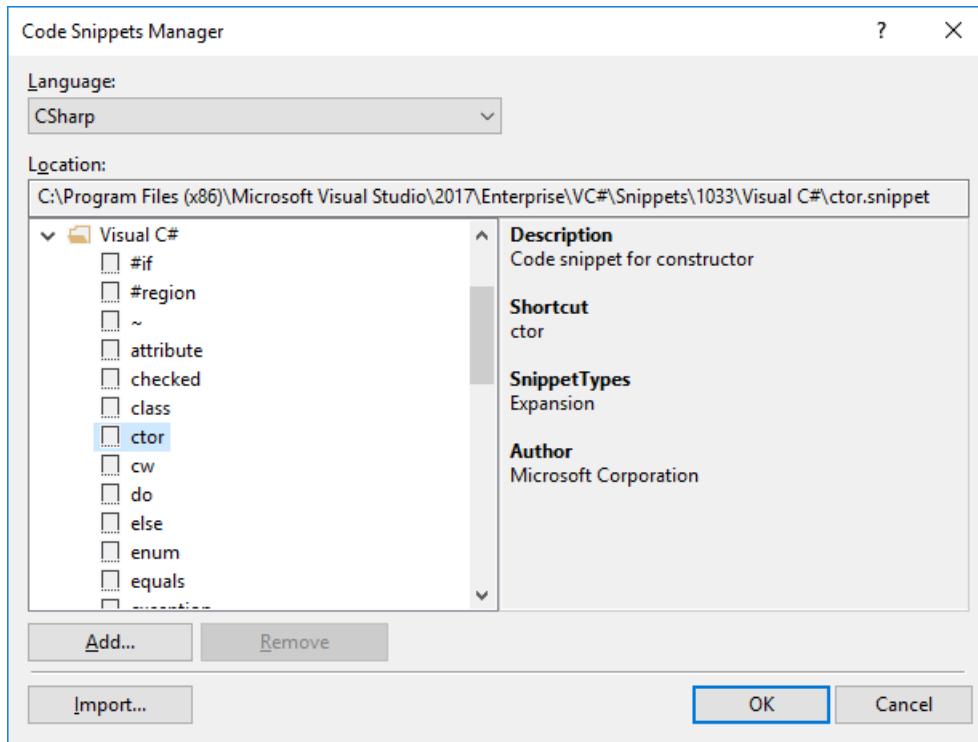
- [Code Snippets](#)
- [Quick Actions](#)
- [Refactoring](#)
- [Code Generation and T4 Text Templates](#)

Code snippets

2/6/2018 • 2 min to read • [Edit Online](#)

Code snippets are small blocks of reusable code that can be inserted in a code file using a context menu command or a combination of hotkeys. They typically contain commonly-used code blocks such as `try-finally` or `if-else` blocks, but they can be used to insert entire classes or methods.

Code snippets are available for a multitude of languages, including C#, C++, Visual Basic, XML, and T-SQL, to name a few. To view all the available installed snippets for a language, open the **Code Snippets Manager** from the **Tools** menu in Visual Studio, and choose the language from the drop-down menu at the top.



Code snippets can be accessed in the following general ways:

- On the menu bar, choose **Edit > IntelliSense > Insert Snippet...**
- From the right-click or context menu in the code editor, choose **Snippet > Insert Snippet...**
- From the keyboard, press **Ctrl+K+X**

Expansion snippets and surround-with snippets

In Visual Studio there are two kinds of code snippet: expansion snippets, which are added at a specified insertion point and may replace a snippet shortcut, and surround-with snippets (C# and C++ only), which are added around a selected block of code.

An example of an expansion snippet: in C# the shortcut `tryf` is used to insert a `try-finally` block:

```
try
{
}
finally
{
}
```

You can insert this snippet by clicking **Insert Snippet** in the context menu of the code window, then **Visual C#**, then type `tryf`, and then press **Tab**. Or, you can type `tryf` and press **Tab** twice.

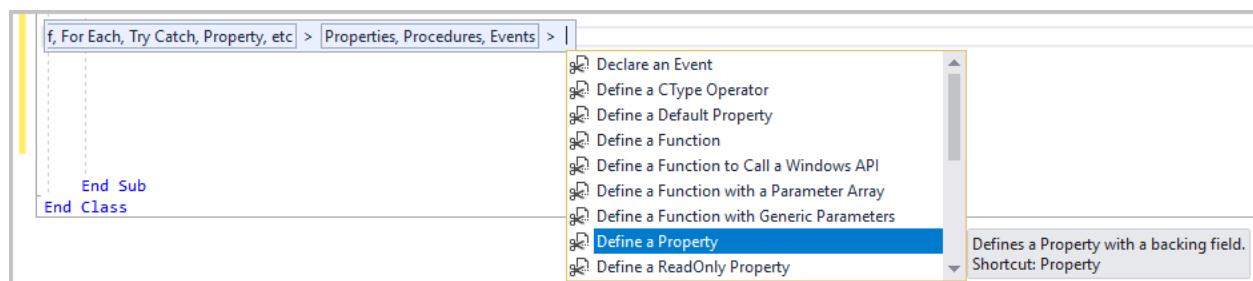
An example of a surround-with snippet: in C++ the shortcut `if` can be used either as an insertion snippet or as a surround-with snippet. If you select a line of code (for example `return FALSE;`), and then choose **Surround With > if**, the snippet is expanded around the line:

```
if (true)
{
    return FALSE;
}
```

Snippet replacement parameters

Snippets can contain replacement parameters, which are placeholders that you must replace to fit the precise code you are writing. In the previous example `true` is a replacement parameter, which you would replace with the appropriate condition. The replacement you make is repeated for every instance of the same replacement parameter in the snippet.

For example, in Visual Basic there is a code snippet that inserts a property. To insert the snippet, choose **Snippet... > Insert Snippet** from the right-click or context menu in a Visual Basic code file. Then, choose **Code Patterns > Properties, Procedures, Events > Define a Property**.



The following code is inserted:

```
Private newPropertyValue As String
Public Property NewProperty() As String
    Get
        Return newPropertyValue
    End Get
    Set(ByVal value As String)
        newPropertyValue = value
    End Set
End Property
```

If you change `newPropertyValue` to `m_property`, then every instance of `newPropertyValue` is changed. If you change `String` to `Int` in the property declaration, then the value in the set method is also changed to `Int`.

See also

[Walkthrough: Creating a code snippet](#)

[How to: Distribute code snippets](#)

[Best practices for Using code snippets](#)

[Troubleshooting snippets](#)

[C# code snippets](#)

[Visual C++ code snippets](#)

[Code snippets schema reference](#)

C# code snippets

2/6/2018 • 4 min to read • [Edit Online](#)

Code snippets are ready-made snippets of code you can quickly insert into your code. For example, the `for` code snippet creates an empty `for` loop. Some code snippets are surround-with code snippets, which enable you to select lines of code, and then choose a code snippet which incorporates the selected lines of code. For example, when you select lines of code and then activate the `for` code snippet, it creates a `for` loop with those lines of code inside the loop block. Code snippets can make writing program code quicker, easier, and more reliable.

You can insert a code snippet at the cursor location, or insert a surround-with code snippet around the currently selected code. The Code Snippet Inserter is invoked through the **Insert Code Snippet** or **Surround With** commands on the **IntelliSense** menu, or by using the keyboard shortcuts **Ctrl+K,X** or **Ctrl+K,S** respectively.

The Code Snippet Inserter displays the code snippet name for all available code snippets. The Code Snippet Inserter also includes an input dialog box where you can type the name of the code snippet, or part of the code snippet name. The Code Snippet Inserter highlights the closest match to a code snippet name. Pressing **Tab** at any time will dismiss the Code Snippet Inserter and insert the currently selected code snippet. Pressing **Esc** or clicking the mouse in the Code Editor will dismiss the Code Snippet Inserter without inserting a code snippet.

Default code snippets

By default the following code snippets are included in Visual Studio for C#.

| NAME (OR SHORTCUT) | DESCRIPTION | VALID LOCATIONS TO INSERT SNIPPET |
|--------------------|---|--|
| #if | Creates a <code>#if</code> directive and a <code>#endif</code> directive. | Anywhere. |
| #region | Creates a <code>#region</code> directive and a <code>#endregion</code> directive. | Anywhere. |
| ~ | Creates a <code>finalizer</code> (destructor) for the containing class. | Inside a class. |
| attribute | Creates a declaration for a class that derives from Attribute . | Inside a namespace (including the global namespace), a class, or a struct. |
| checked | Creates a <code>checked</code> block. | Inside a method, an indexer, a property accessor, or an event accessor. |
| class | Creates a class declaration. | Inside a namespace (including the global namespace), a class, or a struct. |
| ctor | Creates a constructor for the containing class. | Inside a class. |
| cw | Creates a call to WriteLine . | Inside a method, an indexer, a property accessor, or an event accessor. |
| do | Creates a <code>do</code> <code>while</code> loop. | Inside a method, an indexer, a property accessor, or an event accessor. |

| NAME (OR SHORTCUT) | DESCRIPTION | VALID LOCATIONS TO INSERT SNIPPET |
|--------------------|--|--|
| else | Creates an <code>else</code> block. | Inside a method, an indexer, a property accessor, or an event accessor. |
| enum | Creates an <code>enum</code> declaration. | Inside a namespace (including the global namespace), a class, or a struct. |
| equals | Creates a method declaration that overrides the <code>Equals</code> method defined in the <code>Object</code> class. | Inside a class or a struct. |
| exception | Creates a declaration for a class that derives from an exception (<code>Exception</code> by default). | Inside a namespace (including the global namespace), a class, or a struct. |
| for | Creates a <code>for</code> loop. | Inside a method, an indexer, a property accessor, or an event accessor. |
| foreach | Creates a <code>foreach</code> loop. | Inside a method, an indexer, a property accessor, or an event accessor. |
| forr | Creates a <code>for</code> loop that decrements the loop variable after each iteration. | Inside a method, an indexer, a property accessor, or an event accessor. |
| if | Creates an <code>if</code> block. | Inside a method, an indexer, a property accessor, or an event accessor. |
| indexer | Creates an indexer declaration. | Inside a class or a struct. |
| interface | Creates an <code>interface</code> declaration. | Inside a namespace (including the global namespace), a class, or a struct. |
| invoke | Creates a block that safely invokes an event. | Inside a method, an indexer, a property accessor, or an event accessor. |
| iterator | Creates an iterator. | Inside a class or a struct. |
| iterindex | Creates a "named" iterator and indexer pair by using a nested class. | Inside a class or a struct. |
| lock | Creates a <code>lock</code> block. | Inside a method, an indexer, a property accessor, or an event accessor. |
| mbox | Creates a call to <code>System.Windows.Forms.MessageBox.Show</code> . You may have to add a reference to <code>System.Windows.Forms.dll</code> . | Inside a method, an indexer, a property accessor, or an event accessor. |
| namespace | Creates a <code>namespace</code> declaration. | Inside a namespace (including the global namespace). |
| prop | Creates an <code>auto-implemented property</code> declaration. | Inside a class or a struct. |

| NAME (OR SHORTCUT) | DESCRIPTION | VALID LOCATIONS TO INSERT SNIPPET |
|--------------------|---|--|
| propfull | Creates a property declaration with <code>get</code> and <code>set</code> accessors. | Inside a class or a struct. |
| propg | Creates a read-only auto-implemented property with a private <code>set</code> accessor. | Inside a class or a struct. |
| sim | Creates a <code>static int Main</code> method declaration. | Inside a class or a struct. |
| struct | Creates a <code>struct</code> declaration. | Inside a namespace (including the global namespace), a class, or a struct. |
| svm | Creates a <code>static void Main</code> method declaration. | Inside a class or a struct. |
| switch | Creates a <code>switch</code> block. | Inside a method, an indexer, a property accessor, or an event accessor. |
| try | Creates a <code>try-catch</code> block. | Inside a method, an indexer, a property accessor, or an event accessor. |
| tryf | Creates a <code>try-finally</code> block. | Inside a method, an indexer, a property accessor, or an event accessor. |
| unchecked | Creates an <code>unchecked</code> block. | Inside a method, an indexer, a property accessor, or an event accessor. |
| unsafe | Creates an <code>unsafe</code> block. | Inside a method, an indexer, a property accessor, or an event accessor. |
| using | Creates a <code>using</code> directive. | Inside a namespace (including the global namespace). |
| while | Creates a <code>while</code> loop. | Inside a method, an indexer, a property accessor, or an event accessor. |

See also

[Code Snippet Functions](#)

[Code Snippets](#)

[Template Parameters](#)

[How to: Use Surround-with Code Snippets](#)

Visual C++ code snippets

2/6/2018 • 2 min to read • [Edit Online](#)

In Visual Studio, you can use code snippets to add commonly-used code to your C++ code files. In general, you can use code snippets in much the same way as in C#, but the set of default code snippets is different.

You can either add a code snippet at a particular location in your code (insertion) or surround some selected code with a code snippet.

Inserting a code snippet

To insert a code snippet, open a C++ code file (.cpp or .h), click somewhere inside the file, and do one of the following:

- Right-click to get the context menu and select **Insert Snippet**
- In the **Edit / IntelliSense** menu, select **Insert Snippet**
- Use the hotkeys: **Ctrl+K+X**

You should see a list of choices beginning with **#if**. When you select **#if**, you should see the following code added to the file:

```
#if 0  
#endif // 0
```

You can then replace the 0 with the correct condition.

Using a code snippet to surround selected code

To use a code snippet to surround selected code, select a line (or multiple lines) and do one of the following:

- Right-click to get the context menu, and select **Surround With**
- From the **Edit > IntelliSense** menu, select **Surround With**
- Using a keyboard, press: **CTRL+K+S**

Select **#if**. You should see something like this:

```
#if 0  
#include "pch.h" // or whatever line you had selected  
#endif // 0
```

You can then replace the 0 with the correct condition.

Where can I find a complete list of the C++ code snippets?

You can find the complete list of C++ code snippets by going to the **Code Snippets Manager** (on the **Tools** menu) and setting the **Language** to **Visual C++**. In the window below, expand **Visual C++**. You should see the names of all the C++ code snippets in alphabetical order.

The names of most code snippets are self-explanatory, but some names might be confusing.

Class vs. classi

The **class** snippet provides the definition of a class named MyClass, with the appropriate default constructor and destructor, where the definitions of the constructor and destructor are located outside the class:

```
class MyClass
{
public:
    MyClass();
    ~MyClass();

private:
};

MyClass::MyClass()
{
}

MyClass::~MyClass()
{
}
```

The **classi** code snippet also provides the definition of a class named MyClass, but the default constructor and destructor are defined inside the class definition:

```
class MyClass
{
public:
    MyClass()
    {

    ~MyClass()
    {

private:
};

}
```

for vs. forr vs rfor

There are three different **for** snippets that provide different kinds of `for` loops.

The **rfor** snippet provides a [range-based](#) for loop ([link](#)). This construct is preferred over index-based `for` loops.

```
for (auto& i : v)
{



}
```

The **for** snippet provides a `for` loop in which the condition is based on the length (in `size_t`) of an object.

```
for (size_t i = 0; i < length; i++)  
{  
}
```

The **forr** snippet provides a reverse `for` loop in which the condition is based on the length (in integers) of an object.

```
for (int i = length - 1; i >= 0; i--)  
{  
}
```

The destructor snippet (~)

The destructor snippet (~) shows different behavior in different contexts. If you insert this snippet inside a class, it provides a destructor for that class. For example, given the following code:

```
class SomeClass {  
};
```

If you insert the destructor snippet, it provides a destructor for SomeClass:

```
class SomeClass {  
    ~SomeClass()  
    {  
    }  
};
```

If you try to insert the destructor snippet outside a class, it provides a destructor with a placeholder name:

```
~TypeNamePlaceholder()  
{
```

See also

[Code snippets](#)

How to: Insert XML comments for documentation generation

2/8/2018 • 1 min to read • [Edit Online](#)

Visual Studio can help you document code elements such as classes and methods, by automatically generating the standard XML documentation comment structure. At compile time, you can generate an XML file that contains the documentation comments. The compiler-generated XML file can be distributed alongside your .NET assembly so that Visual Studio and other IDEs can use IntelliSense to show quick information about types and members.

Additionally, the XML file can be run through tools like [DocFX](#) and [Sandcastle](#) to generate API reference websites.

NOTE

The **Insert Comment** command that automatically inserts XML documentation comments is available in [C#](#) and [Visual Basic](#). However, you can manually insert [XML documentation comments in C++](#) files and still generate XML documentation files at compile time.

To insert XML comments for a code element

1. Place your text cursor above the element you want to document, for example, a method.

2. Do one of the following:

- Type `///` in C#, or `''` in Visual Basic
- From the **Edit** menu, choose **IntelliSense > Insert Comment**
- From the right-click or context menu on or just above the code element, choose **Snippet > Insert Comment**

The XML template is immediately generated above the code element. For example, when commenting a method, it generates the `<summary>` element, a `<param>` element for each parameter, and a `<returns>` element to document the return value.

```
/// <summary>
/// |
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
public string GetUsername(int id)
{
    return "username";
}

''' <summary>
''' |
''' </summary>
''' <param name="id"></param>
''' <returns></returns>
Public Function GetUsername(id As Integer) As String
    Return "username"
End Function
```

3. Enter descriptions for each XML element to fully document the code element.

```
/// <summary>
/// Gets the username associated with the specified ID.
/// </summary>
/// <param name="id">The unique user ID.</param>
/// <returns>A string containing the username for the ID specified.</returns>
public string GetUsername(int id)
{
    return "username";
}
```

NOTE

There is an [option](#) to toggle XML documentation comments after typing `///` in C# or `'''` Visual Basic. From the menu bar, choose **Tools** > **Options** to open the **Options** dialog box. Then, navigate to **Text Editor** > **C#** or **Basic** > **Advanced**. In the **Editor Help** section, look for the **Generate XML documentation comments** option.

See also

[XML documentation comments \(C# Programming Guide\)](#)

[Documenting your code with XML comments \(C# Guide\)](#)

[How to: Create XML documentation \(Visual Basic\)](#)

[C++ Comments](#)

[XML Documentation \(C++\)](#)

[Code generation](#)

How to: Use surround-with code snippets

2/6/2018 • 1 min to read • [Edit Online](#)

The following procedures describe how to use surround-with code snippets. Surround-with code snippets are available three ways: through a keyboard shortcut, through the **Edit** menu, and through the context menu.

To use surround-with code snippets through keyboard shortcut

1. In the Visual Studio IDE, open the file that you intend to edit.
2. In the Code Editor, select text to surround.
3. Type **Ctrl+K, Ctrl+S**.
4. Select the code snippet from the code snippet list using the mouse, or by typing the name of the code snippet and pressing **Tab** or **Enter**.

To use surround-with code snippets through the Edit menu

1. In the Visual Studio IDE, open the file that you intend to edit.
2. In the Code Editor, select text to surround.
3. From the **Edit** menu, select **IntelliSense** and then select the **Surround With** command.
4. Select the code snippet from the code snippet inserter and then press **Tab** or **Enter**.

Alternatively, you can type the name of the code snippet, and then press **Tab** or **Enter**.

To use surround-with code snippets through the context menu

1. In the Visual Studio IDE, open the file that you intend to edit.
2. In the Code Editor, select text to surround.
3. Right-click the selected text and then select the **Surround With** command from the context menu.
4. Select the code snippet from the code snippet inserter and then press **Tab** or **Enter**.

Alternatively, you can type the name of the code snippet, and then press **Tab** or **Enter**.

See also

- [C# Code Snippets](#)
- [Code Snippet Picker](#)

Best practices for using code snippets

2/6/2018 • 2 min to read • [Edit Online](#)

The code in a code snippet shows only the most basic way to do something. For most applications, the code must be modified to suit the application.

Handling exceptions

Typically, code snippet Try...Catch blocks catch and rethrow all exceptions. That may not be the right choice for your project. For each exception, there are several ways to respond. For examples, see [How to: Handle an Exception Using try/catch \(C#\)](#) and [Try...Catch...Finally Statement \(Visual Basic\)](#).

File locations

When you adapt file locations to your application, you should think about the following:

- Finding an accessible location. Users may not have access to the Program Files folder of the computer, so storing files with the application files may not work.
- Finding a secure location. Storing files in the root folder (C:\) is not secure. For application data, we recommend the *Application Data* folder. For individual user data, the application can create a file for each user in the *Documents* folder.
- Using a valid file name. You can use the [OpenFileDialog](#) and [SaveFileDialog](#) controls to reduce the likelihood of invalid file names. Be aware that between the time the user selects a file and the time your code manipulates the file, the file may be deleted. In addition, the user may not have permissions to write to the file.

Security

How secure a snippet is depends on where it is used in the source code and how it is modified once it is in the code. The following list contains a few of the areas that must be considered.

- File and database access
- Code access security
- Protecting resources (such as event logs, registry)
- Storing secrets
- Verifying inputs
- Passing data to scripting technologies

For more information, see [Securing Applications](#).

Downloaded code snippets

IntelliSense code snippets installed by Visual Studio are not in themselves a security hazard. However, they can create security risks in your application. Snippets downloaded from the Internet should be treated like any other downloaded content - with extreme caution.

- Download snippets only from sites you trust, and use up-to-date virus software.

- Open all downloaded snippet files in Notepad or the XML editor of Visual Studio and review them carefully before installing them. Look for the following issues:
 - The snippet code could damage your system if you execute it. Read the source code carefully before running it.
 - The Help URL block of the snippet file can contain URLs that execute a malicious script file or display an offensive Web site.
 - The snippet may contain references that are added silently to your project and may be loaded from anywhere on your system. These references may have been downloaded to your computer from where you downloaded the snippet. The snippet may then make a call to a method in the reference that executes malicious code. To protect yourself against such an attack, review the Imports and References blocks of the snippet file.

See also

[Visual Basic IntelliSense Code Snippets](#)

[Securing Applications](#)

[Code Snippets](#)

Walkthrough: Creating a Code Snippet

12/22/2017 • 5 min to read • [Edit Online](#)

You can create a code snippet with only a few steps. All you need to do is create an XML file, fill in the appropriate elements, and add your code to it. You can also add references and replacement parameters to your code. You can add the snippet to your Visual Studio installation by using the Import button on the Code Snippets Manager ([Tools, Code Snippets Manager...](#)).

Snippet Template

The following is the basic snippet template:

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets
    xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
    <CodeSnippet Format="1.0.0">
        <Header>
            <Title></Title>
        </Header>
        <Snippet>
            <Code Language="">
                <![CDATA[]]>
            </Code>
        </Snippet>
    </CodeSnippet>
</CodeSnippets>
```

To Create a Code Snippet

1. Create a new XML file in Visual Studio and add the template shown above.
2. Fill in the title of the snippet, e.g. "Hello World VB", in the Title element.
3. Fill in the language of the snippet in the Languages attribute of the Code element. For this example, use "VB".
4. Add some code in the CDATA section inside the Code element, for example:

```
<Code Language="VB">
    <![CDATA[Console.WriteLine("Hello, World!")]]>
</Code>
```

5. Save the snippet as VBCodeSnippet.snippet.

To Add a Code Snippet to Visual Studio

1. You can add your own snippets to your Visual Studio installation by using the Code Snippets Manager. Open the Code Snippets Manager ([Tools, Code Snippets Manager...](#)).
2. Click the **Import** button.
3. Go to the location where you saved the code snippet in the previous procedure, select it, and click **Open**.
4. The **Import Code Snippet** dialog opens, asking you to choose where to add the snippet from the choices in the right pane. One of the choices should be **My Code Snippets**. Select it and click **Finish**, then **OK**.

5. The snippet is copied to the following location:

%USERPROFILE%\Documents\Visual Studio 2017\Code Snippets\Visual Basic\My Code Snippets

6. Test your snippet by opening a Visual Basic project and opening a code file. In the file choose **Snippets**, **Insert Snippet** from the context menu, then **My Code Snippets**. You should see a snippet named **My Visual Basic Code Snippet**. Double-click it.

`Console.WriteLine("Hello, World!")` is inserted in the code file.

Adding Description and Shortcut Fields

1. Description fields give more information about your code snippet when viewed in the Code Snippets Manager. The shortcut is a tag that users can type in order to insert your snippet. Edit the snippet you have added by opening the file %USERPROFILE%\Documents\Visual Studio 2017\Code Snippets\Visual Basic\My Code Snippet\VBCodeSnippet.snippet.
2. Add Author and Description elements to the Header element, and fill them in.
3. The Header element should look something like this:

```
<Header>
  <Title>Hello World VB</Title>
  <Author>Myself</Author>
  <Description>Says Hello to the world.</Description>
</Header>
```

4. Open the Code Snippets Manager and select your code snippet. In the right pane you should see that the Description and Author fields are now populated.
5. To add a shortcut, add a Shortcut element alongside the Author and Description element:

```
<Header>
  <Title>Hello World VB</Title>
  <Author>Myself</Author>
  <Description>Says Hello to the world.</Description>
  <Shortcut>hello</Shortcut>
</Header>
```

6. Save the snippet file again.
7. To test the shortcut, open a Visual Basic project and open a code file. Type `hello` in the file and press **Tab** twice.

The snippet code is inserted.

To Add References and Imports

1. You can add a reference to a project by using the References element, and add an Imports declaration by using the Imports element. (This works for C# as well.) For example, if you change `Console.WriteLine` in the code example to `MessageBox.Show`, you may need to add the System.Windows.Forms.dll assembly to the project.
2. Open your snippet.
3. Add the References element under the Snippet element:

```
<References>
  <Reference>
    <Assembly>System.Windows.Forms.dll</Assembly>
  </Reference>
</References>
```

4. Add the Imports element under the Snippet element:

```
<Imports>
  <Import>
    <Namespace>System.Windows.Forms</Namespace>
  </Import>
</Imports>
```

5. Change the CDATA section to the following:

```
<![CDATA[MessageBox.Show("Hello, World!")]]>
```

6. Save the snippet.
7. Open a Visual Basic project and add the snippet.
8. You will see an Imports statement at the top of the code file:

```
Imports System.Windows.Forms
```

9. Look at the project's properties. The References tab includes a reference to System.Windows.Forms.dll.

Adding Replacements

1. You may want parts of your code snippets to be replaced by the user, for example if you add a variable and want the user to replace the variable with one in the current project. You can provide two types of replacements: literals and objects. Literals are strings of some type (string literals, variable names, or string representations of numeric values). Objects are instances of some type other than a string. In this procedure you will declare a literal replacement and an object replacement, and change the code to reference these replacements.
2. Open your snippet.
3. This example uses a SQL connection string, so you need to change the Imports and References elements to add the appropriate references:

```

<References>
    <Reference>
        <Assembly>System.Data.dll</Assembly>
    </Reference>
    <Reference>
        <Assembly>System.Xml.dll</Assembly>
    </Reference>
</References>
<Imports>
    <Import>
        <Namespace>System.Data</Namespace>
    </Import>
    <Import>
        <Namespace>System.Data.SqlClient</Namespace>
    </Import>
</Imports>

```

4. To declare a literal replacement for the SQL connection string, add a Declarations element under the Snippet element, and in it add a Literal element with subelements for the ID, the tooltip, and the default value for the replacement:

```

<Declarations>
    <Literal>
        <ID>SqlConnString</ID>
        <ToolTip>Replace with a SQL connection string.</ToolTip>
        <Default>"SQL connection string"</Default>
    </Literal>
</Declarations>

```

5. To declare an object replacement for the SQL connection, add an Object element inside the Declarations element, and add sub-elements for the ID, the type of the object, the tooltip, and the default value. The resulting Declarations element should look like this:

```

<Declarations>
    <Literal>
        <ID>SqlConnString</ID>
        <ToolTip>Replace with a SQL connection string.</ToolTip>
        <Default>"SQL connection string"</Default>
    </Literal>
    <Object>
        <ID>SqlConnection</ID>
        <Type>System.Data.SqlClient.SqlConnection</Type>
        <ToolTip>Replace with a connection object in your application.</ToolTip>
        <Default>dcConnection</Default>
    </Object>
</Declarations>

```

6. In the code section, you reference the replacements with surrounding \$ signs, for example `$replacement$`:

```

<Code Language="VB" Kind="method body">
<![CDATA[Dim daCustomers As SqlDataAdapter
    Dim selectCommand As SqlCommand

    daCustomers = New SqlCommand.SqlDataAdapter()
    selectCommand = new SqlCommand.SqlCommand($SqlConnString$)
    daCustomers.SelectCommand = selectCommand
    daCustomers.SelectCommand.Connection = $SqlConnection$]]>
</Code>

```

7. Save the snippet.

8. Open a Visual Basic project and add the snippet.
9. The code should look like the following, where the replacements `SQL connection string` and `dcConnection` are highlighted in light orange. Choose **Tab** to navigate from one to the other.

```
Dim daCustomers As SqlDataAdapter
Dim selectCommand As SqlCommand

daCustomers = New SqlClient.SqlDataAdapter()
selectCommand = New SqlClient.SqlCommand("SQL connection string")
daCustomers.SelectCommand = selectCommand
daCustomers.SelectCommand.Connection = dcConnection
```

See Also

[Code Snippets Schema Reference](#)

How to: Distribute Code Snippets

12/22/2017 • 3 min to read • [Edit Online](#)

You can simply give your code snippets to your friends and have them install the snippets on their own computers by using the Code Snippets Manager. However, if you have several snippets to distribute or would like to distribute them more widely, you include your snippet file in a Visual Studio extension, which Visual Studio users can install.

You must install the Visual Studio SDK in order to create Visual Studio extensions. Find the version of the VSSDK that matches your Visual Studio installation at [Visual Studio Downloads](#).

Setting up the Extension

In this procedure we will use the same Hello World code snippet created in [Walkthrough: Creating a Code Snippet](#). We will supply the .snippet text, so you don't have to go back and make one.

1. Create a new VSIX project named **TestSnippet**. ([File](#), [New](#), [Project](#), **Visual C# (or Visual Basic)**, **Extensibility**.)
2. In the **TestSnippet** project, add a new XML file and call it **VBCodeSnippet.snippet**. Replace the content with the following:

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets
    xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
    <CodeSnippet Format="1.0.0">
        <Header>
            <Title>Hello World VB</Title>
            <Shortcut>HelloWorld</Shortcut>
            <Description>Inserts code</Description>
            <Author>MSIT</Author>
            <SnippetTypes>
                <SnippetType>Expansion</SnippetType>
                <SnippetType>SurroundsWith</SnippetType>
            </SnippetTypes>
        </Header>
        <Snippet>
            <Code Language="VB">
                <![CDATA[Console.WriteLine("Hello, World!")]]>
            </Code>
        </Snippet>
    </CodeSnippet>
</CodeSnippets>
```

Setting up the Directory Structure

1. In the Solution Explorer, select the project node and add a folder that has the name you want the snippet to have in the Code Snippets Manager. In this case it should be **HelloWorldVB**.
2. Move the .snippet file to the **HelloWorldVB** folder.
3. Select the .snippet file in the Solution Explorer, and in the **Properties** window make sure **Build Action** is set to **Content, Copy to Output Directory** is set to **Copy always**, and **Include in VSIX** is set to **true**.

Adding the .pkgdef file

1. Add a text file to the **HelloWorldVB** folder and name it **HelloWorldVB.pkgdef**. This file is used to add certain keys to the registry. In this case it adds a new key to

`HKCU\Software\Microsoft\VisualStudio\15.0\Languages\CodeExpansions\Basic`.

2. Add the following lines to the file.

```
// Visual Basic  
[$RootKey$\Languages\CodeExpansions\Basic\Paths]  
"HelloWorldVB"="$PackageFolder$"
```

If you examine this key, you can see how to specify different languages.

3. Select the .pkgdef file in the Solution Explorer, and in the **Properties** window make sure **Build Action** is set to **Content, Copy to Output Directory** is set to **Copy always**, and **Include in VSIX** is set to **true**.
4. Add the .pkgdef file as an asset in the VSIX manifest. In the source.extension.vsixmanifest file, go to the **Assets** tab and click **New**.
5. In the **Add New Asset** dialog, set the **Type** to **Microsoft.VisualStudio.VsPackage**, the **Source** to **File on filesystem**, and the **Path** to **HelloWorldVB.pkgdef** (which should appear in the dropdown).

Testing the Snippet

1. Now you can make sure that the code snippet works in the experimental instance of Visual Studio. The experimental instance is a second copy of Visual Studio that is separate from the one you use to write code. It allows you to work on an extension without affecting your development environment.
2. Build the project and start debugging. A second instance of Visual Studio should appear.
3. In the experimental instance, go to **Tools / Code Snippets Manager** and set the **Language** to **Basic**. You should see HelloWorldVB as one of the folders, and you should be able to expand the folder to see the HelloWorldVB snippet.
4. Test the snippet. In the experimental instance, open a Visual Basic project and open one of the code files. Place your cursor somewhere in the code, right-click, and on the context menu select **Insert Snippet**.
5. You should see HelloWorldVB as one of the folders. Double-click it. You should see a pop-up **Insert Snippet: HelloWorldVB >** that has a dropdown **HelloWorldVB**. Click the HelloWorldVB dropdown. You should see the following line added to the file:

```
Console.WriteLine("Hello, World!")
```

See Also

[Code Snippets](#)

Code Snippet functions

3/12/2018 • 1 min to read • [Edit Online](#)

There are three functions available to use with C# code snippets. Functions are specified in the [Function](#) element of the code snippet. For information on creating code snippets, see [Code Snippets](#).

Functions

The following table describes the functions available for use with the [Function](#) element in code snippets.

| FUNCTION | DESCRIPTION | LANGUAGE |
|--|---|-----------|
| <code>GenerateSwitchCases(EnumerationLiteral)</code> | Generates a switch statement and a set of case statements for the members of the enumeration specified by the <code>EnumerationLiteral</code> parameter. The <code>EnumerationLiteral</code> parameter must be either a reference to an enumeration literal or an enumeration type. | Visual C# |
| <code>ClassName()</code> | Returns the name of the class that contains the inserted snippet. | Visual C# |
| <code>SimpleTypeName(TypeName)</code> | Reduces the <code>TypeName</code> parameter to its simplest form in the context in which the snippet was invoked. | Visual C# |

Example

The following example shows how to use the `GenerateSwitchCases` function. When this snippet is inserted and an enumeration is entered into the `$switch_on$` literal, the `$cases$` literal generates a `case` statement for every value in the enumeration.

```
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>switch</Title>
      <Shortcut>switch</Shortcut>
      <Description>Code snippet for switch statement</Description>
      <Author>Microsoft Corporation</Author>
      <SnippetTypes>
        <SnippetType>Expansion</SnippetType>
      </SnippetTypes>
    </Header>
    <Snippet>
      <Declarations>
        <Literal>
          <ID>expression</ID>
          <ToolTip>Expression to switch on</ToolTip>
          <Default>switch_on</Default>
        </Literal>
        <Literal Editable="false">
          <ID>cases</ID>
          <Function>GenerateSwitchCases($expression$)</Function>
          <Default>default:</Default>
        </Literal>
      </Declarations>
      <Code Language="csharp">
        <![CDATA[
          switch ($expression$)
          {
            $cases$
          }
        ]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

Example

The following example shows how to use the `className` function. When this snippet is inserted, the `$classname$` literal is replaced with the name of the enclosing class at that location in the code file.

```

<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>Common constructor pattern</Title>
      <Shortcut>ctor</Shortcut>
      <Description>Code Snippet for a constructor</Description>
      <Author>Microsoft Corporation</Author>
      <SnippetTypes>
        <SnippetType>Expansion</SnippetType>
      </SnippetTypes>
    </Header>
    <Snippet>
      <Declarations>
        <Literal>
          <ID>type</ID>
          <Default>int</Default>
        </Literal>
        <Literal>
          <ID>name</ID>
          <Default>field</Default>
        </Literal>
        <Literal default="true" Editable="false">
          <ID>classname</ID>
          <ToolTip>Class name</ToolTip>
          <Function>ClassName()</Function>
          <Default>ClassNamePlaceholder</Default>
        </Literal>
      </Declarations>
      <Code Language="csharp" Format="CData">
        <![CDATA[
          public $classname$ ($type$ $name$)
          {
            this._$name$ = $name$;
          }
          private $type$ _$name$;
        ]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>

```

Example

This example shows how to use the `SimpleTypeName` function. When this snippet is inserted into a code file, the `$SystemConsole$` literal will be replaced with the simplest form of the `Console` type in the context in which the snippet was invoked.

```
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>Console_WriteLine</Title>
      <Shortcut>cw</Shortcut>
      <Description>Code snippet for Console.WriteLine</Description>
      <Author>Microsoft Corporation</Author>
      <SnippetTypes>
        <SnippetType>Expansion</SnippetType>
      </SnippetTypes>
    </Header>
    <Snippet>
      <Declarations>
        <Literal Editable="false">
          <ID>SystemConsole</ID>
          <Function>SimpleTypeName(global::System.Console)</Function>
        </Literal>
      </Declarations>
      <Code Language="csharp">
        <![CDATA[
          $SystemConsole$.WriteLine();
        ]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

See also

[Function Element](#)

[Code Snippets Schema Reference](#)

Code Snippets Schema Reference

1/26/2018 • 17 min to read • [Edit Online](#)

IntelliSense Code Snippets are pre-authored pieces of code that are ready to be inserted into your application with Visual Studio. You can increase productivity by providing code snippets that reduce the amount of time spent typing repetitive code or searching for samples. You can use the IntelliSense Code Snippet XML schema to create your own code snippets and add them to the code snippets that Visual Studio already includes.

IntelliSense Code Snippets Schema Elements

| | | |
|----------------------|-------------------|----------------------|
| Assembly Element | HelpUrl Element | References Element |
| Author Element | ID Element | Shortcut Element |
| Code Element | Import Element | Snippet Element |
| CodeSnippet Element | Imports Element | SnippetType Element |
| CodeSnippets Element | Keyword Element | SnippetTypes Element |
| Declarations Element | Keywords Element | Title Element |
| Default Element | Literal Element | ToolTip Element |
| Description Element | Namespace Element | Type Element |
| Function Element | Object Element | Url Element |
| Header Element | Reference Element | |

Assembly Element

Specifies the name of the assembly referenced by the code snippet.

The text value of the **Assembly** element is either the friendly text name of the assembly, such as `System.dll`, or its strong name, such as `System,Version=1.0.0.1,Culture=neutral,PublicKeyToken=9b35aa323c18d4fb1`.

```
<Assembly>
  AssemblyName
</Assembly>
```

| PARENT ELEMENT | DESCRIPTION |
|-------------------|--|
| Reference Element | Contains information about assembly references required by the code snippet. |

A text value is required. This text specifies the assembly that the code snippet references.

Author Element

Specifies the name of the snippet author. The **Code Snippets Manager** displays the name stored in the `Author` element of the code snippet.

```
<Author>
  Code Snippet Author
</Author>
```

| PARENT ELEMENT | DESCRIPTION |
|----------------|--|
| Header Element | Contains general information about the code snippet. |

A text value is required. This text specifies the author of the code snippet.

Code Element

Provides a container for short code blocks.

Keywords

Two reserved words are available for use in the text of the `Code` element: `end` and `$selected$`. `end` marks the location to place the cursor after the code snippet is inserted. `$selected$` represents text selected in the document that is to be inserted into the snippet when it is invoked. For example, given a snippet that includes:

```
$selected$ is a great color.
```

If the word "Blue" is selected when the user invokes the template, the result is:

```
Blue is a great color.
```

You may not use either `end` or `$selected$` more than once in a code snippet. If you do, only the second instance is recognized. Given a snippet that includes:

```
$selected$ is a great color. I love $selected$.
```

If the word "Blue" is selected, the result is:

```
is a great color. I love Blue.
```

The initial space appears because there is a space between `$selected$` and `is`.

All other `$` keywords are dynamically defined in the `<Literal>` and `<Object>` tags.

Following is the structure of the Code element:

```
<Code Language="Language"
      Kind="method body/method decl/type decl/page/file/any"
      Delimiter="Delimiter">
  Code to insert
</Code>
```

A text value is required. This text specifies the code, along with the literals and objects, that you can use when this

code snippet is inserted into a code file.

Attributes

There are three attributes available for the Code element:

- **Language** - *Required* attribute that specifies the language of the code snippet. The value can be one of the following:

| VALUE | DESCRIPTION |
|------------|---|
| VB | Identifies a Visual Basic code snippet. |
| CSharp | Identifies a C# code snippet. |
| CPP | Identifies a C++ code snippet. |
| XML | Identifies an XML code snippet. |
| JavaScript | Identifies a JavaScript code snippet. |
| SQL | Identifies a SQL code snippet. |
| HTML | Identifies an HTML code snippet. |

- **Kind** - *Optional* attribute that specifies the kind of code that the snippet contains, and the location at which a code snippet must be inserted for the code snippet to compile. The value can be one of the following:

| VALUE | DESCRIPTION |
|-------------|---|
| method body | Specifies that the code snippet is a method body, and therefore, must be inserted inside a method declaration. |
| method decl | Specifies that the code snippet is a method, and therefore, must be inserted inside a class or module. |
| type decl | Specifies that the code snippet is a type, and therefore, must be inserted inside a class, module, or namespace. |
| file | Specifies that the snippet is a full code file. These code snippets can be inserted alone into a code file, or inside a namespace. |
| any | Specifies that the snippet can be inserted anywhere. This tag is used for code snippets that are context-independent, such as comments. |

- **Delimiter** - *Optional* attribute that specifies the delimiter used to describe literals and objects in the code.

By default, the delimiter is `$`.

Parent element

| PARENT ELEMENT | DESCRIPTION |
|-----------------|--|
| Snippet Element | Contains the references, imports, declarations, and code for the code snippet. |

CodeSnippet Element

Allows you to specify a heading and multiple IntelliSense Code Snippets, which you can insert into Visual Studio code files.

```
<CodeSnippet Format="x.x.x">
  <Header>... </Header>
  <Snippet>... </Snippet>
</CodeSnippet>
```

| ATTRIBUTE | DESCRIPTION |
|-----------------------------------|---|
| <code>Format</code> | Required attribute. Specifies the schema version of the code snippet. The Format attribute must be a string in the syntax of x.x.x, where each "x" represents a numerical value of the version number. Visual Studio will ignore code snippets with <code>Format</code> attributes that it does not understand. |
| CHILD ELEMENT | DESCRIPTION |
| <code>Header Element</code> | Required element. Contains general information about the code snippet. There must be exactly one <code>Header</code> element in a code snippet. |
| <code>Snippet Element</code> | Required element. Contains the code that will be inserted by Visual Studio. There must be exactly one <code>Snippet</code> element in a code snippet. |
| PARENT ELEMENT | DESCRIPTION |
| <code>CodeSnippets Element</code> | Root element of the code snippet XML schema. |

CodeSnippets Element

Groups `CodeSnippet Element` elements. The `CodeSnippets` element is the root element of the code snippet XML schema.

| | |
|---|---|
| <pre><CodeSnippets> <CodeSnippet>... </CodeSnippet> </CodeSnippets></pre> | |
| <code>CodeSnippet Element</code> | Optional element. Parent element for all code snippet data. There may be zero or more <code>CodeSnippet</code> elements in a <code>CodeSnippets</code> element. |

Declarations Element

Specifies the literals and objects that make up the parts of a code snippet that you can edit.

```
<Declarations>
  <Literal>... </Literal>
  <Object>... </Object>
</Declarations>
```

| CHILD ELEMENT | DESCRIPTION |
|-----------------|---|
| Literal Element | Optional element. Defines the literals of the code snippet that you can edit. There may be zero or more <code>Literal</code> elements in a <code>Declarations</code> element. |
| Object Element | Optional element. Defines the objects of the code snippet that you can edit. There may be zero or more <code>Object</code> elements in a <code>Declarations</code> element. |
| PARENT ELEMENT | DESCRIPTION |
| Snippet Element | Contains the references, imports, declarations, and code for the code snippet. |

Default Element

Specifies the default value of the literal or object for an IntelliSense Code Snippet.

```
<Default>
  Default value
</Default>
```

| PARENT ELEMENT | DESCRIPTION |
|-----------------|---|
| Literal Element | Defines the literal fields of the code snippet that you can edit. |
| Object Element | Defines the object fields of the code snippet that you can edit. |

A text value is required. This text specifies the default value of the literal or object that fills the fields of the code snippet that you can edit.

Description Element

Specifies descriptive information about the contents of an IntelliSense Code Snippet.

```
<Description>
  Code Snippet Description
</Description>
```

| PARENT ELEMENT | DESCRIPTION |
|----------------|--|
| Header Element | Contains general information about the code snippet. |

A text value is required. This text describes the code snippet.

Function Element

Specifies a function to execute when the literal or object receives focus in Visual Studio.

NOTE

The `Function` element is only supported in C# code snippets.

```
<Function>
  FunctionName
</Function>
```

| PARENT ELEMENT | DESCRIPTION |
|-----------------|---|
| Literal Element | Defines the literal fields of the code snippet that you can edit. |
| Object Element | Defines the object fields of the code snippet that you can edit. |

A text value is required. This text specifies a function to execute when the literal or object field receives focus in Visual Studio.

Specifies general information about the IntelliSense Code Snippet.

```
<Header>
  <Title>... </Title>
  <Author>... </Author>
  <Description>... </Description>
  <HelpUrl>... </HelpUrl>
  <SnippetTypes>... </SnippetTypes>
  <Keywords>... </Keywords>
  <Shortcut>... </Shortcut>
</Header>
```

| CHILD ELEMENT | DESCRIPTION |
|---------------------|--|
| Author Element | Optional element. The name of the person or company that authored the code snippet. There may be zero or one <code>Author</code> elements in a <code>Header</code> element. |
| Description Element | Optional element. A description of the code snippet. There may be zero or one <code>Description</code> elements in a <code>Header</code> element. |
| HelpUrl Element | Optional element. A URL that contains more information about the code snippet. There may be zero or one <code>HelpURL</code> elements in a <code>Header</code> element. Note: Visual Studio does not use the <code>HelpUrl</code> element. The element is part of the IntelliSense Code Snippet XML schema and any code snippet containing the element will validate, but the value of the element is never used. |
| Keywords Element | Optional element. Groups <code>Keyword</code> elements. There may be zero or one <code>Keywords</code> elements in a <code>Header</code> element. |

| CHILD ELEMENT | DESCRIPTION |
|----------------------|---|
| Shortcut Element | Optional element. Specifies the shortcut text that can be used to insert the snippet. There may be zero or one <code>Shortcut</code> elements in a <code>Header</code> element. |
| SnippetTypes Element | Optional element. Groups <code>SnippetType</code> elements. There may be zero or one <code>SnippetTypes</code> elements in a <code>Header</code> element. If there are no <code>SnippetTypes</code> elements, the code snippet is always valid. |
| Title Element | Required element. The friendly name of the code snippet. There must be exactly one <code>Title</code> element in a <code>Header</code> element. |
| PARENT ELEMENT | DESCRIPTION |
| CodeSnippet Element | Parent element for all code snippet data. |

HelpUrl Element

Specifies a URL that provides more information about a code snippet.

NOTE

Visual Studio does not use the `HelpUrl` element. The element is part of the IntelliSense Code Snippet XML schema and any code snippet containing the element will validate, but the value of the element is never used.

```
<HelpUrl>
    www.microsoft.com
</HelpUrl>
```

| PARENT ELEMENT | DESCRIPTION |
|----------------|--|
| Header Element | Contains general information about the code snippet. |

A text value is optional. This text specifies the URL to visit for more information about a code snippet.

ID Element

Specifies a unique identifier for a `Literal` or `Object` element. No two literals or objects in the same code snippet can have the same text value in their `ID` elements. Literals and objects cannot contain an `ID` element with a value of end. The value `end` is reserved, and is used to mark the location to place the cursor after the code snippet is inserted.

```
<ID>
    Unique Identifier
</ID>
```

| PARENT ELEMENT | DESCRIPTION |
|-----------------|---|
| Literal Element | Defines the literal fields of the code snippet that you can edit. |
| Object Element | Defines the object fields of the code snippet that you can edit. |

A text value is required. This text specifies the unique identifier for the object or literal.

Import Element

Specifies the imported namespaces used by an IntelliSense Code Snippet.

NOTE

The `Import` element is only supported for Visual Basic projects.

```
<Import>
  <Namespace>... </Namespace>
</Import>
```

| CHILD ELEMENT | DESCRIPTION |
|-------------------|---|
| Namespace Element | Required element. Specifies the namespace used by the code snippet. There must be exactly one <code>Namespace</code> element in an <code>Import</code> element. |

| PARENT ELEMENT | DESCRIPTION |
|-----------------|--|
| Imports Element | Grouping element for <code>Import</code> elements. |

Imports Element

Groups individual `Import` elements.

NOTE

The `Imports` element is only supported for Visual Basic projects.

```
<Imports>
  <Import>... </Import>
<Imports>
```

| CHILD ELEMENT | DESCRIPTION |
|----------------|---|
| Import Element | Optional element. Contains the imported namespaces for the code snippet. There may be zero or more <code>Import</code> elements in an <code>Imports</code> element. |

| PARENT ELEMENT | DESCRIPTION |
|-----------------|--|
| Snippet Element | Contains the references, imports, declarations, and code for the code snippet. |

Keyword Element

Specifies a custom keyword for the code snippet. The code snippet keywords are used by Visual Studio and represent a standard way for online content providers to add custom keywords for searching or categorization.

```
<Keyword>
  Code Snippet Keyword
</Keyword>
```

| PARENT ELEMENT | DESCRIPTION |
|------------------|--|
| Keywords Element | Groups individual <code>Keyword</code> elements. |

A text value is required. The keyword for the code snippet.

Keywords Element

Groups individual `Keyword` elements. The code snippet keywords are used by Visual Studio and represent a standard way for online content providers to add custom keywords for searching or categorization

```
<Keywords>
  <Keyword>... </Keyword>
  <Keyword>... </Keyword>
<Keywords>
```

| CHILD ELEMENT | DESCRIPTION |
|-----------------|--|
| Keyword Element | Optional element. Contains individual keywords for the code snippet. There may be zero or more <code>Keyword</code> elements in a <code>Keywords</code> element. |
| PARENT ELEMENT | DESCRIPTION |
| Header Element | Contains general information about the code snippet. |

Literal Element

Defines the literals of the code snippet that you can edit. The `Literal` element is used to identify a replacement for a piece of code that is entirely contained within the snippet, but will likely be customized after it is inserted into the code. For example, literal strings, numeric values, and some variable names should be declared as literals.

Literals and objects cannot contain an **ID** element with a value of selected or end. The value `$selected$` represents text selected in the document that is to be inserted into the snippet when it is invoked. `end` marks the location to place the cursor after the code snippet is inserted.

```

<Literal Editable="true/false">
  <ID>... </ID>
  <ToolTip>... </ToolTip>
  <Default>... </Default>
  <Function>... </Function>
</Literal>

```

| ATTRIBUTE | DESCRIPTION |
|-----------------------------------|--|
| <code>Editable</code> | Optional <code>Boolean</code> attribute. Specifies whether or not you can edit the literal after the code snippet is inserted. The default value of this attribute is <code>true</code> . |
| CHILD ELEMENT | DESCRIPTION |
| <code>Default Element</code> | Required element. Specifies the literal's default value when you insert the code snippet. There must be exactly one <code>Default</code> element in a <code>Literal</code> element. |
| <code>Function Element</code> | Optional element. Specifies a function to execute when the literal receives focus in Visual Studio. There may be zero or one <code>Function</code> elements in a <code>Literal</code> element. |
| <code>ID Element</code> | Required element. Specifies a unique identifier for the literal. There must be exactly one <code>ID</code> element in a <code>Literal</code> element. |
| <code>ToolTip Element</code> | Optional element. Describes the expected value and usage of the literal. There may be zero or one <code>Tooltip</code> elements in a <code>Literal</code> element. |
| PARENT ELEMENT | DESCRIPTION |
| <code>Declarations Element</code> | Contains the literals and objects of a code snippet that you can edit. |

Namespace Element

Specifies the namespace that must be imported for the code snippet to compile and run. The namespace specified in the `Namespace` element is automatically added to an `Imports` statement at the beginning of the code, if it does not already exist.

NOTE

The `Namespace` element is only supported for Visual Basic projects.

```

<Namespace>
  Namespace
</Namespace>

```

| PARENT ELEMENT | DESCRIPTION |
|----------------|----------------------------------|
| Import Element | Imports the specified namespace. |

A text value is required. This text specifies a namespace that the code snippet assumes is imported.

Object Element

Defines the objects of the code snippet that you can edit. The `Object` element is used to identify an item that is required by the code snippet but is likely to be defined outside of the snippet itself. For example, Windows Forms controls, ASP.NET controls, object instances, and type instances should be declared as objects. Object declarations require that a type be specified, which is done with the `Type` element.

```
<Object Editable="true/false">
  <ID>... </ID>
  <Type>... </Type>
  <ToolTip>... </ToolTip>
  <Default>... </Default>
  <Function>... </Function>
</Object>
```

| ATTRIBUTE | DESCRIPTION |
|-----------------------|--|
| <code>Editable</code> | Optional <code>Boolean</code> attribute. Specifies whether or not you can edit the literal after the code snippet is inserted. The default value of this attribute is <code>true</code> . |
| CHILD ELEMENT | DESCRIPTION |
| Default Element | Required element. Specifies the literal's default value when you insert the code snippet. There must be exactly one <code>Default</code> element in a <code>Literal</code> element. |
| Function Element | Optional element. Specifies a function to execute when the literal receives focus in Visual Studio. There may be zero or one <code>Function</code> elements in a <code>Literal</code> element. |
| ID Element | Required element. Specifies a unique identifier for the literal. There must be exactly one <code>ID</code> element in a <code>Literal</code> element. |
| ToolTip Element | Optional element. Describes the expected value and usage of the literal. There may be zero or one <code>Tooltip</code> elements in a <code>Literal</code> element. |
| Type Element | Required element. Specifies the type of the object. There must be exactly one <code>Type</code> element in an <code>Object</code> element. |
| PARENT ELEMENT | DESCRIPTION |
| Declarations Element | Contains the literals and objects of a code snippet that you can edit. |

Reference Element

Specifies information about the assembly references required by the code snippet.

```
<Reference>
  <Assembly>... </Assembly>
  <Url>... </Url>
</Reference>
```

| CHILD ELEMENT | DESCRIPTION |
|--------------------|--|
| Assembly Element | Required element. Contains the name of the assembly referenced by the code snippet. There must be exactly one <code>Assembly</code> element in a <code>Reference</code> element. |
| Url Element | Optional element. Contains a URL that provides more information about the referenced assembly. There may be zero or one <code>Url</code> elements in a <code>Reference</code> element. |
| PARENT ELEMENT | DESCRIPTION |
| References Element | Grouping element for <code>Reference</code> elements. |

References Element

Groups individual `Reference` elements.

```
<References>
  <Reference>... </Reference>
</References>
```

| CHILD ELEMENT | DESCRIPTION |
|-------------------|--|
| Reference Element | Optional element. Contains information about assembly references for the code snippet. There may be zero or more <code>Reference</code> elements in a <code>References</code> element. |
| PARENT ELEMENT | DESCRIPTION |
| Snippet Element | Contains the references, imports, declarations, and code for the code snippet. |

Shortcut Element

Specifies the shortcut text used to insert the snippet. The text value of a `Shortcut` element can only contain alphanumeric characters, hyphens (-), and underscores (_).

Caution

_ and - are not supported characters in C++ snippet shortcuts.

```
<Shortcut>
  Shortcut Text
</Shortcut>
```

| PARENT ELEMENT | DESCRIPTION |
|----------------|--|
| Header Element | Contains general information about the code snippet. |

A text value is optional. This text is used as a shortcut for inserting the code snippet.

Snippet Element

Specifies the references, imports, declarations, and code for the code snippet.

```
<Snippet>
  <References>... </References>
  <Imports>... </Imports>
  <Declarations>... </Declarations>
  <Code>... </Code>
</Snippet>
```

| CHILD ELEMENT | DESCRIPTION |
|----------------------|---|
| Code Element | Required element. Specifies the code that you want to insert into a documentation file. There must be exactly one <code>Code</code> element in a <code>Snippet</code> element. |
| Declarations Element | Optional element. Specifies the literals and objects that make up the parts of a code snippet that you can edit. There may be zero or one <code>Declarations</code> elements in a <code>Snippet</code> element. |
| Imports Element | Optional element. Groups individual <code>Import</code> elements. There may be zero or one <code>Imports</code> elements in a <code>Snippet</code> element. |
| References Element | Optional element. Groups individual <code>Reference</code> elements. There may be zero or one <code>References</code> elements in a <code>Snippet</code> element. |

| PARENT ELEMENT | DESCRIPTION |
|---------------------|--|
| CodeSnippet Element | Allows you to specify a heading and multiple IntelliSense Code Snippets, which you can insert into Visual Studio code files. |

SnippetType Element

Specifies how Visual Studio inserts the code snippet.

```
<SnippetType>
  SurroundsWith/Expansion
<SnippetType>
```

| PARENT ELEMENT | DESCRIPTION |
|----------------------|---|
| SnippetTypes Element | Groups <code>SnippetType</code> elements. |

The text value must be one of the following values:

- `SurroundsWith` : allows the code snippet to be placed around a selected piece of code.
- `Expansion` : allows the code snippet to be inserted at the cursor.
- `Refactoring` : specifies that the code snippet is used during C# refactoring. `Refactoring` cannot be used in custom code snippets.

SnippetTypes Element

Groups individual `SnippetType` elements. If the `SnippetTypes` element is not present, the code snippet can be inserted anywhere in the code.

```
<SnippetTypes>
  <SnippetType>... </SnippetType>
  <SnippetType>... </SnippetType>
<SnippetTypes>
```

| CHILD ELEMENT | DESCRIPTION |
|---------------------|---|
| SnippetType Element | Optional element. Specifies how Visual Studio inserts the code snippet into the code. There may be zero or more <code>SnippetType</code> elements in a <code>SnippetTypes</code> element. |
| PARENT ELEMENT | DESCRIPTION |
| Header Element | Specifies general information about the code snippet. |

Title Element

Specifies the title for the code snippet. The title stored in the `Title` element of the code snippet appears in the **Code Snippet Picker** and in the code snippet's description in the **Code Snippets Manager**.

```
<Title>
  Code Snippet Title
<Title>
```

| PARENT ELEMENT | DESCRIPTION |
|----------------|---|
| Header Element | Specifies general information about the code snippet. |

A text value is required. This text specifies the title of the code snippet.

ToolTip Element

Describes the expected value and usage of a literal or object in a code snippet, which Visual Studio displays in a ToolTip when it inserts the code snippet into a project. The ToolTip text is displayed when the mouse hovers over the literal or object after the code snippet has been inserted.

```
<ToolTip>
    ToolTip description
</ToolTip>
```

| PARENT ELEMENT | DESCRIPTION |
|-----------------|---|
| Literal Element | Defines the literal fields of the code snippet that you can edit. |
| Object Element | Defines the object fields of the code snippet that you can edit. |

A text value is required. This text specifies the ToolTip description to be associated with the object or literal in the code snippet.

Type Element

Specifies the type of the object. The `object` element is used to identify an item that is required by the code snippet but is likely to be defined outside of the snippet itself. For example, Windows Forms controls, ASP.NET controls, object instances, and type instances should be declared as objects. Object declarations require that a type be specified, which is done with the `Type` element.

```
<Type>
    Type
</Type>
```

| PARENT ELEMENT | DESCRIPTION |
|----------------|--|
| Object Element | Defines the object fields of the code snippet that you can edit. |

A text value is required. This text specifies the type of the object.

Url Element

Specifies a URL that provides more information about the referenced assembly.

NOTE

The `url` element is only supported for Visual Basic projects.

```
<Url>
    www.microsoft.com
</Url>
```

| PARENT ELEMENT | DESCRIPTION |
|-------------------|---|
| Reference Element | Specifies the assembly references required by the code snippet. |

A text value is required. This text specifies a URL with more information about the referenced assembly. This URL is displayed when the reference cannot be added to the project.

See Also

[Code Snippets](#)

[Walkthrough: Creating a Code Snippet](#)

Troubleshooting Snippets

12/22/2017 • 1 min to read • [Edit Online](#)

Problems with IntelliSense code snippets are typically caused by two problems: a corrupt snippet file or bad content in the snippet file.

Common Problems

The Snippet Cannot Be Dragged from File Explorer to a Visual Studio Source File

- The XML in the snippet file may be corrupt. The **XML Editor** in Visual Studio can locate problems in the XML structure.
- The snippet file may not conform to the snippet schema. The **XML Editor** in Visual Studio can locate problems in the XML structure.

The Code Has Compiler Errors That Are Not Highlighted

- You may be missing a project reference. Examine the documentation about the snippet. If the reference is not found on the computer, you will need to install it. Inserting a snippet should add to the project any references needed. If the snippet is missing the reference information, that can be reported to the snippet creator as an error.
- A variable may be undefined. Undefined variables in a snippet should be highlighted. If not, that can be reported to the snippet creator as an error.

See Also

[Code Snippets](#)

Quick Actions

1/13/2018 • 1 min to read • [Edit Online](#)

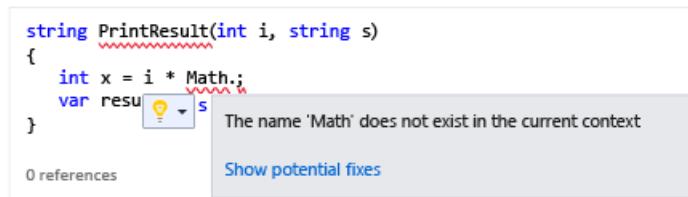
Quick Actions let you easily refactor, generate, or otherwise modify code with a single action. Quick Actions are available for C#, C++, and Visual Basic code files. Some actions are specific to a language, and others apply to all languages.

Quick Actions can be applied by using the light bulb icon , or by pressing **Ctrl+.** when your cursor is on the appropriate line of code. You will see a light bulb if there is a red squiggle and Visual Studio has a suggestion for how to fix the issue. For instance if you have an error indicated by a red squiggle, a light bulb will appear when fixes are available for that error.

For any language, third parties can provide custom diagnostics and suggestions, for example as part of an SDK, and Visual Studio light bulbs will light up based on those rules.

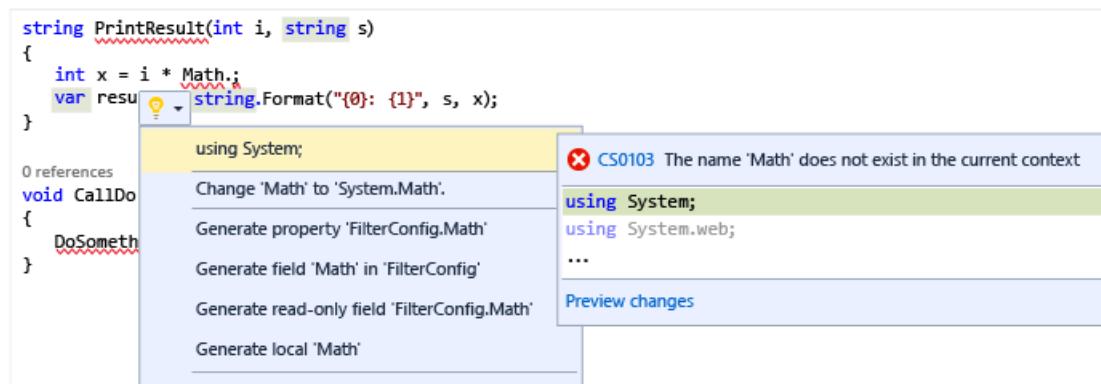
To see a light bulb

1. In many cases, light bulbs spontaneously appear when you hover the mouse at the point of an error, or in the left margin of the editor when you move the caret into a line that has an error in it. When you see a red squiggle, you can hover over it to display the light bulb. You can also cause a light bulb to display when you use the mouse or keyboard to go to anywhere in the line where the issue occurs.
2. Press **Ctrl+.** anywhere on a line to invoke the light bulb and go directly to the list of potential fixes.



To see potential fixes

Either click on the down arrow or the Show potential fixes link to display a list of quick actions that the light bulb can take for you.



See also

[Code generation in Visual Studio](#)

[Common Quick Actions](#)

[Code styles and Quick Actions](#)

Writing and refactoring code (C++)

Common Quick Actions

2/6/2018 • 13 min to read • [Edit Online](#)

The sections in this topic list some of the common Quick Actions that are applicable to both C# and Visual Basic code.

Actions that fix errors

Correct misspelled symbol or keyword

If you accidentally misspell a type or keyword in Visual Studio, this Quick Action will automatically correct it for you. You'll see these items in the light bulb menu as "**Change 'misspelled word' to 'correct word'**". For example:

```
// Before
private viod MyMethod()
{
}

// Change 'viod' to 'void'

// After
private void MyMethod()
{}
```

```
' Before
Function MyFunction as Intger
End Function

' Change 'Intger' to 'Integer'

' After
Function MyFunction as Integer
End Function
```

| ERROR ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|-----------------|----------------------|-----------------------------|
| CS0103, BC30002 | C# and Visual Basic | Visual Studio 2015 Update 2 |

Resolve git merge conflict

These Quick Actions enable you to resolve git merge conflicts by "taking a change", which removes the conflicting code and markers.

```

// Before
private void MyMethod()
{
    <<<<< HEAD
    if (true)
    {

    }
    =====
    if (false)
    {

    }
    >>>>> upstream
}

// Take changes from 'HEAD'

// After
private void MyMethod()
{
    if (true)
    {

    }
}

```

| ERROR ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|-----------------|----------------------|---------------------------------|
| CS8300, BC37284 | C# and Visual Basic | Visual Studio 2017 version 15.3 |

Make method synchronous

When using the `async` or `Async` keyword on a method, it is expected that somewhere inside that method the `await` or `Await` keyword will also be used. However, if this isn't the case, a Quick Action will appear that will allow you to make the method synchronous by removing the `async` or `Async` keyword and changing the return type. Use the **Make method synchronous** option from the Quick Actions menu.

```

// Before
async Task<int> MyAsyncMethod()
{
    return 3;
}

// Make method synchronous

// After
int MyAsyncMethod()
{
    return 3;
}

```

```

' Before
Async Function MyAsyncMethod() As Task(Of Integer)
    Return 3
End Function

' Make method synchronous

' After
Function MyAsyncMethod() As Integer
    Return 3
End Function

```

| ERROR ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|-----------------|----------------------|-----------------------------|
| CS1998, BC42356 | C# and Visual Basic | Visual Studio 2015 Update 2 |

Make method asynchronous

When using the `await` or `Await` keyword inside of a method, it is expected that the method itself is marked with the `async` or `Async` keyword. However, if this isn't the case, a Quick Action will appear that will allow you to make the method asynchronous. Use the **Make method/Function asynchronous** option from the Quick Actions menu.

```

// Before
int MyAsyncMethod()
{
    return await Task.Run(...);
}

// Make method asynchronous

// After
async Task<int> MyAsyncMethod()
{
    return await Task.Run(...);
}

```

```

' Before
Function MyAsyncMethod() as Integer
    Return Await Task.Run(...)
End Function

' Make method asynchronous

' After
Async Function MyAsyncMethod() As Task(Of Integer)
    Return Await Task.Run(...)
End Function

```

| ERROR ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|-----------------|----------------------|--------------------|
| CS4032, BC37057 | C# and Visual Basic | Visual Studio 2017 |

Actions that remove unnecessary code

Remove unnecessary usings/Imports

The **Remove Unnecessary Usings/Imports** Quick Action will remove any unused `using` and `Import` statements for the current file. When you select this item, unused namespace imports will be immediately removed.

| APPLICABLE LANGUAGES | SUPPORTED VERSION |
|----------------------|------------------------|
| C# and Visual Basic | Visual Studio 2015 RTW |

Remove unnecessary cast

If you cast a type to another type which doesn't require a cast, the **Remove Unnecessary Cast** Quick Action item will remove the cast from your code.

```
// before
int number = (int)3;

// Remove Unnecessary Cast

// after
int number = 3;
```

```
' Before
Dim number as Integer = CType(3, Integer)

' Remove Unnecessary Cast

' After
Dim number as Integer = 3
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|------------------------|
| IDE0004 | C# and Visual Basic | Visual Studio 2015 RTW |

Remove unused variables

This Quick Action enables you to remove variables that have been declared but never used in your code.

```
// Before
public MyMethod()
{
    var unused = 8;
    var used = 1;
    return DoStuff(used);
}

// Remove unused variables

// After
public MyMethod()
{
    var used = 1;
    return DoStuff(used);
}
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|-----------------|----------------------|---------------------------------|
| CS0219, BC42024 | C# and Visual Basic | Visual Studio 2017 version 15.3 |

Remove type from default value expression

This Quick Action removes the value type from a default value expression and uses the [default literal](#) when the compiler can infer the type of the expression.

```
// Before
void DoWork(CancellationToken cancellationToken = default(CancellationToken)) { ... }

// Simplify default expression

// After
void DoWork(CancellationToken cancellationToken = default) { ... }
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|---------------------------------|
| IDE0034 | C# 7.1+ | Visual Studio 2017 version 15.3 |

Actions that add missing code

Add usings/Imports for types in reference assemblies, NuGet packages, or other types in your solution

Using types located in other projects in your solution will display the Quick Action automatically, however the others need to be enabled from the **Tools > Options > C#** or **Basic > Advanced** tab:

- Suggest usings/imports for types in reference assemblies
- Suggest usings/imports for types in NuGet packages

When enabled, if you use a type in a namespace that is currently not imported, but exists in a reference assembly or NuGet package, the using/import statement will be created.

```
// Before
Debug.WriteLine("Hello");

// using System.Diagnostics;

// After
using System.Diagnostics;

Debug.WriteLine("Hello");
```

```
' Before
Debug.WriteLine("Hello")

' Imports System.Diagnostics

// After
Imports System.Diagnostics

Debug.WriteLine("Hello")
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|-----------------|----------------------|-----------------------------|
| CS0103, BC30451 | C# and Visual Basic | Visual Studio 2015 Update 2 |

Add missing cases/default case/both

When creating a `switch` statement in C#, or `Select Case` statement in Visual Basic, you can use a Code Action to automatically add missing case items, a default case statement, or both.

Consider the following enumeration and empty `switch` or `Select Case` statement:

```

enum MyEnum
{
    Item1,
    Item2,
    Item3
}

...

MyEnum myEnum = MyEnum.Item1;

switch(myEnum)
{
}

```

```

Enum MyEnum
    Item1
    Item2
    Item3
End Enum

...

Dim myEnum as MyEnum = MyEnum.Item1

Select Case myEnum
End Select

```

Using the **Add Both** Quick Action fills in missing cases and adds a default case:

```

switch(myEnum)
{
    case MyEnum.Item1:
        break;
    case MyEnum.Item2:
        break;
    case MyEnum.Item3:
        break;
    default:
        break;
}

```

```

Select Case myEnum
    Case MyEnum.Item1
        Exit Select
    Case MyEnum.Item2
        Exit Select
    Case Else
        Exit Select
End Select

```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|---------------------------------|
| IDE0010 | C# and Visual Basic | Visual Studio 2017 version 15.3 |

Add null checks for parameters

This Quick Action enables you to add a check in your code to tell whether a parameter is null.

```

// Before
class MyClass
{
    public string MyProperty { get; set; }

    public MyClass(string myProperty) // cursor inside myProperty
    {
        MyProperty = myProperty;
    }
}

// Add null check

// After
class MyClass
{
    public string MyProperty { get; set; }

    public MyClass(string myProperty)
    {
        MyProperty = myProperty ?? throw new ArgumentNullException(nameof(myProperty));
    }
}

```

APPLICABLE LANGUAGES

SUPPORTED VERSION

C# and Visual Basic

Visual Studio 2017 version 15.3

Add argument name

```

// Before
var date = new DateTime(1997, 7, 8);

// Include argument name 'year' (include trailing arguments)

// After
var date = new DateTime(year: 1997, month: 7, day: 8);

```

APPLICABLE LANGUAGES

SUPPORTED VERSION

C# and Visual Basic

Visual Studio 2017 version 15.3

Add braces

The Add braces Quick Action wraps braces around single-line `if` statements.

```

// Before
if (true)
    return "hello,world";

// Add braces

// After
if (true)
{
    return "hello,world";
}

```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|------------------------|
| IDE0011 | C# | Visual Studio 2017 RTW |

Add and order modifiers

These Quick Actions help organize modifiers by enabling you to sort existing and add missing accessibility modifiers.

```
// Before
enum Color
{
    Red, White, Blue
}

// Add accessibility modifiers

// After
internal enum Color
{
    Red, White, Blue
}
```

```
// Before
static private int thisFieldIsPublic;

// Order modifiers

// After
private static int thisFieldIsPublic;
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|---------------------------------|
| IDE0036 | C# and Visual Basic | Visual Studio 2017 version 15.5 |
| IDE0040 | C# and Visual Basic | Visual Studio 2017 version 15.5 |

Code transformations

Convert if construct to switch

This Quick Action enables you to convert an **if-then-else** construct to a **switch** construct.

```

// Before
if (obj is string s)
{
    Console.WriteLine("obj is a string: " + s);
}

else if (obj is int i && i > 10)
{
    Console.WriteLine("obj is an int greater than 10");
}

// Convert to switch

// After
switch (obj)
{
    case string s:
        Console.WriteLine("Obj is a string: " + s);
        break;
    case int i when i > 10:
        Console.WriteLine("obj is an int greater than 10");
        break;
}

```

```

' Before
If TypeOf obj Is String s Then
    Console.WriteLine("obj is a string: " + s)
Else If TypeOf obj Is Integer i And i > 10 Then
    Console.WriteLine("obj is an int greater than 10")
End If

' Convert to switch

' After
Select Case obj
    Case String s
        Console.WriteLine("Obj is a string: " + s)
        Exit Sub
    Case Integer i when i > 10
        Console.WriteLine("obj is an int greater than 10")
        Exit Sub
End Select

```

| APPLICABLE LANGUAGES | SUPPORTED VERSION |
|----------------------|---------------------------------|
| C# and Visual Basic | Visual Studio 2017 version 15.3 |

Convert to interpolated string

[Interpolated strings](#) are an easy way to express strings with embedded variables, similar to the [String.Format](#) method. This Quick Action recognizes cases where strings are concatenated, or using [String.Format](#), and changes the usage to an interpolated string.

```
// Before
int num = 3;
string s = string.Format("My string with {0} in the middle", num);

// Convert to interpolated string

// After
int num = 3;
string s = $"My string with {num} in the middle";
```

```
' Before
Dim num as Integer = 3
Dim s as String = String.Format("My string with {0} in the middle", num)

' Convert to interpolated string

' After
Dim num as Integer = 3
Dim s As String = $"My string with {num} in the middle"
```

| APPLICABLE LANGUAGES | SUPPORTED VERSION |
|------------------------------|------------------------|
| C# 6.0+ and Visual Basic 14+ | Visual Studio 2017 RTW |

Use object initializers

This Quick Action enables you to use [object initializers](#) rather than invoking the constructor and having additional lines of assignment statements.

```
// Before
var c = new Customer();
c.Age = 21;

// Object initialization can be simplified

// After
var c = new Customer() { Age = 21 };
```

```
' Before
Dim c = New Customer()
c.Age = 21

' Object initialization can be simplified

' After
Dim c = New Customer() With {.Age = 21}
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|------------------------|
| IDE0017 | C# and Visual Basic | Visual Studio 2017 RTW |

Use collection initializers

This Quick Action lets you use [collection initializers](#) rather than multiple calls to the `Add` method of your class.

```
// Before
var list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);

// Collection initialization can be simplified

// After
var list = new List<int> { 1, 2, 3 };
```

```
' Before
Dim list = New List(Of Integer)
list.Add(1)
list.Add(2)
list.Add(3)

' Collection initialization can be simplified

' After
Dim list = New List(Of Integer) From {1, 2, 3}
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|------------------------|
| IDE0028 | C# and Visual Basic | Visual Studio 2017 RTW |

Convert auto property to full property

This Quick Action enables you to convert an auto property to a full property, and vice versa.

```
// Before
private int MyProperty { get; set; }

// Convert to full property

// After
private int MyProperty
{
    get { return _myProperty; }
    set { _myProperty = value; }
}
```

```
' Before
Public Property Name As String

' Convert to full property

' After
Private _Name As String

Public Property Name As String
    Get
        Return _Name
    End Get
    Set
        _Name = Value
    End Set
End Property
```

| APPLICABLE LANGUAGES | SUPPORTED VERSION |
|----------------------|---------------------------------|
| C# and Visual Basic | Visual Studio 2017 version 15.5 |

Convert block body to expression-bodied member

This Quick Action allows you to convert block bodies into expression-bodied members for methods, constructors, operators, properties, indexers, and accessors.

```
//Before
class MyClass4
{
    private int _myProperty;

    public int MyProperty
    {
        get { return _myProperty; }
        set
        {
            _myProperty = value;
        }
    }

    public MyClass4(int myProperty)
    {
        MyProperty = myProperty;
    }

    public void PrintProperty()
    {
        Console.WriteLine(MyProperty);
    }
}

// Use expression body for accessors/constructors/methods

// After
class MyClass4
{
    private int _myProperty;

    public int MyProperty
    {
        get => _myProperty;
        set => _myProperty = value;
    }

    public MyClass4(int myProperty) => MyProperty = myProperty;

    public void PrintProperty() => Console.WriteLine(MyProperty);
}
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|------------------------|
| IDE0021-27 | C# 6.0+ | Visual Studio 2017 RTW |

Convert anonymous function to local function

This Quick Action converts anonymous functions into local functions.

```

// Before
Func<int, int> fibonacci = null;
fibonacci = (int n) =>
{
    return n <= 1 ? 1 : fibonacci(n - 1) + fibonacci(n - 2);
};

// Use local function

// After
int fibonacci(int n)
{
    return n <= 1 ? 1 : fibonacci(n-1) + fibonacci(n-2);
}

```

Convert `ReferenceEquals` **to** `is null`

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|---------------------------------|
| IDE0041 | C# 7.0+ | Visual Studio 2017 version 15.5 |

This Quick Action suggests the use of [pattern matching](#) rather than the `ReferenceEquals` coding-pattern, where possible.

```

// Before
var value = "someString";
if (object.ReferenceEquals(value, null))
{
    return;
}

// Use 'is null' check

// After
var value = "someString";
if (value is null)
{
    return;
}

```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|---------------------------------|
| IDE0039 | C# 7.0+ | Visual Studio 2017 version 15.5 |

Introduce pattern matching

This Quick Action suggests the use of [pattern matching](#) with casts and null checks in C#.

```
// Before
if (o is int)
{
    var i = (int)o;
    ...
}

// Use pattern matching
```

```
// After
if (o is int i)
{
    ...
}
```

```
// Before
var s = o as string;
if (s != null)
{
    ...
}

// Use pattern matching
```

```
// After
if (o is string s)
{
    ...
}
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|------------------------|
| IDE0020 | C# 7.0+ | Visual Studio 2017 RTW |
| IDE0019 | C# 7.0+ | Visual Studio 2017 RTW |

Change base for numeric literals

This Quick Action enables you to convert a numeric literal from one base numeric system to another. For example, you can change a number to hexadecimal or to binary format.

```
// Before
int countdown = 2097152;

// Convert to hex

// After
int countdown = 0x200000;
```

```
' Before
Dim countdown As Integer = 2097152

' Convert to hex

' After
Dim countdown As Integer = &H200000
```

| APPLICABLE LANGUAGES | SUPPORTED VERSION |
|------------------------------|---------------------------------|
| C# 7.0+ and Visual Basic 14+ | Visual Studio 2017 version 15.3 |

Insert digit separators into literals

This Quick Action enables you to add separator characters into literal values.

```
// Before
int countdown = 1000000;

// Separate thousands

// After
int countdown = 1_000_000;
```

```
' Before
Dim countdown As Integer = 1000000

' Separate thousands

' After
Dim countdown As Integer = 1_000_000
```

| APPLICABLE LANGUAGES | SUPPORTED VERSION |
|------------------------------|---------------------------------|
| C# 7.0+ and Visual Basic 14+ | Visual Studio 2017 version 15.3 |

Use explicit tuple names

This Quick Action identifies areas where the explicit tuple name can be used rather than Item1, Item2, etc.

```
// Before
(string name, int age) customer = GetCustomer();
var name = customer.Item1;

// Use explicit tuple name

// After
(string name, int age) customer = GetCustomer();
var name = customer.name;
```

```
' Before
Dim customer As (name As String, age As Integer) = GetCustomer()
Dim name = customer.Item1

' Use explicit tuple name

' After
Dim customer As (name As String, age As Integer) = GetCustomer()
Dim name = customer.name
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|------------------------------|------------------------|
| IDE0033 | C# 7.0+ and Visual Basic 15+ | Visual Studio 2017 RTW |

Use inferred names

These Quick Actions point out when users can use inferred member names in anonymous types or use C# 7.1's inferred tuple element names.

```
// Before
var anon = new { age = age, name = name };

// Use inferred member name

// After
var anon = new { age, name };
```

```
// Before
var tuple = (age: age, name: name);

// Use inferred tuple element name

// After
var tuple = (age, name);
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|----------------------------|
| IDE0037 | C# | Visual Studio 2017 v. 15.5 |
| IDE0037 | C# 7.1+ | Visual Studio 2017 v. 15.5 |

Deconstruct tuple declaration

This Quick Action enables you to deconstruct tuple variable declarations.

```
// Before
var person = GetPersonTuple();
Console.WriteLine($"{person.name} {person.age}");

(int x, int y) point = GetPointTuple();
Console.WriteLine($"{point.x} {point.y}");

// Deconstruct variable declaration

// After
var (name, age) = GetPersonTuple();
Console.WriteLine($"{name} {age}");

(int x, int y) = GetPointTuple();
Console.WriteLine($"{x} {y}");
```

| DIAGNOSTIC ID | APPLICABLE LANGUAGES | SUPPORTED VERSION |
|---------------|----------------------|----------------------------|
| IDE0042 | C# 7.0+ | Visual Studio 2017 v. 15.5 |

See also

[Quick Actions](#)

Generate a class or type in Visual Studio

2/28/2018 • 1 min to read • [Edit Online](#)

This code generation applies to:

- C#
- Visual Basic

What: Lets you immediately generate the code for a class or type.

When: You introduce a new class or type and want to properly declare it, automatically.

Why: You could declare the class or type before using it, however this feature will generate the class or type automatically.

How-to

1. Place your cursor on the line where there is a red squiggle. The red squiggle indicates a class that doesn't yet exist.

- C#:

```
static void Main(string[] args)
{
    MyNewType t = new MyNewType();
}
```

- Visual Basic:

```
Sub Main()
    Dim t As New MyNewType
End Sub
```

2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.

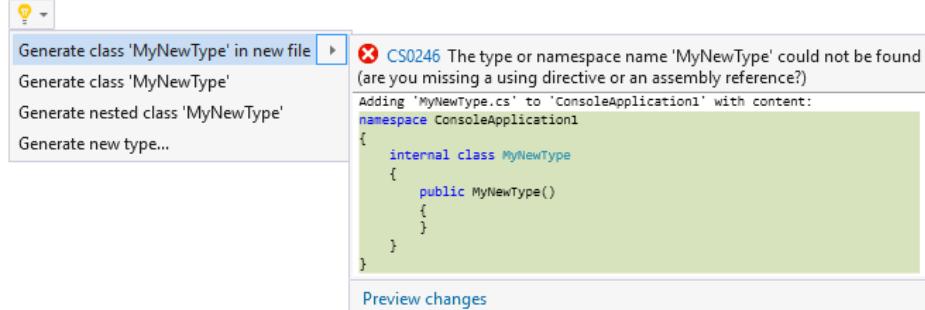
- **Mouse**

- Right-click and select the **Quick Actions and Refactorings** menu.

- Hover over the red squiggle and click the icon which appears.

- Click the icon which appears in the left margin if the text cursor is already on the line with the red squiggle.

```
static void Main(string[] args)
{
    MyNewType t = new MyNewType();
}
```



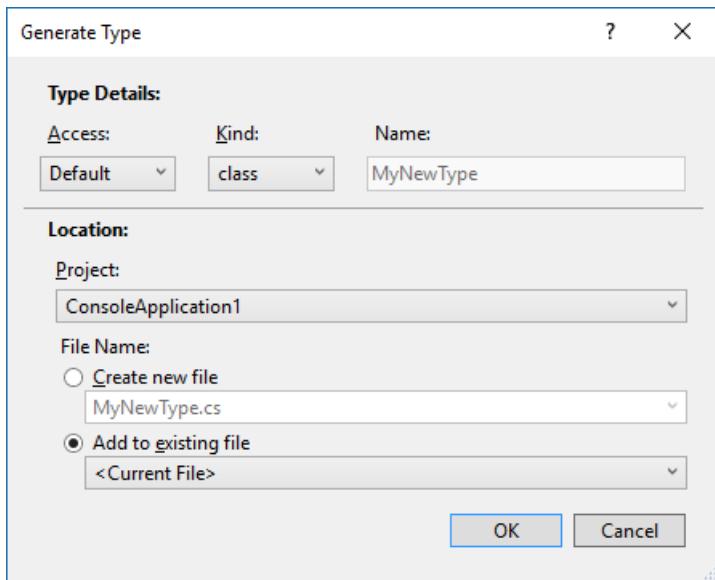
3. Select one of the options from the drop-down menu:

- Generate class '*TypeName*' in new file—Creates a class named *TypeName* in a file named *TypeName.cs/vb*
- Generate class '*TypeName*'—Creates a class named *TypeName* in the current file.
- Generate nested class '*TypeName*'—Creates a class named *TypeName* nested inside the current class.
- Generate new type...—Creates a new class or struct with all of the properties you specify.

TIP

Use the **Preview changes** link at the bottom of the preview window to see all of the changes that will be made before making your selection.

4. If you selected the **Generate new type...** item, the **Generate Type** dialog box opens. Configure the accessibility, kind, and location of the new type.



| SELECTION | DESCRIPTION |
|-----------|--|
| Access | Set the type to have <i>Default</i> , <i>Internal</i> or <i>Public</i> access. |
| Kind | This can be set as <i>class</i> or <i>struct</i> . |
| Name | This cannot be changed and will be the name you already typed. |
| Project | If there are multiple projects in your solution, you can choose where you want the class/struct to live. |
| File Name | You can create a new file or you can add the type to an existing file. |

The class or struct is created. For C#, a constructor is also created.

- C#

```
class MyNewType
{
    public MyNewType()
    {
    }
}
```

- Visual Basic

```
Friend Class MyNewType
End Class
```

See also

- [Code Generation](#)
- [Preview Changes](#)

Generate a method in Visual Studio

2/28/2018 • 1 min to read • [Edit Online](#)

This code generation applies to:

- C#
- Visual Basic

What: Lets you immediately add a method to a class.

When: You introduce a new method and want to properly declare it, automatically.

Why: You could declare the method and parameters before using it, however this feature will generate the declaration automatically.

How-to

1. Place your cursor on the line where there is a red squiggle. The red squiggle indicates a method that doesn't exist yet.

- C#:

```
static void Main(string[] args)
{
    MyNewType t = new MyNewType();
    t.MyMethod(12345);
}
```

- Visual Basic:

```
Sub Main()
    Dim t As New MyNewType
    t.MyMethod(12345)
End Sub
```

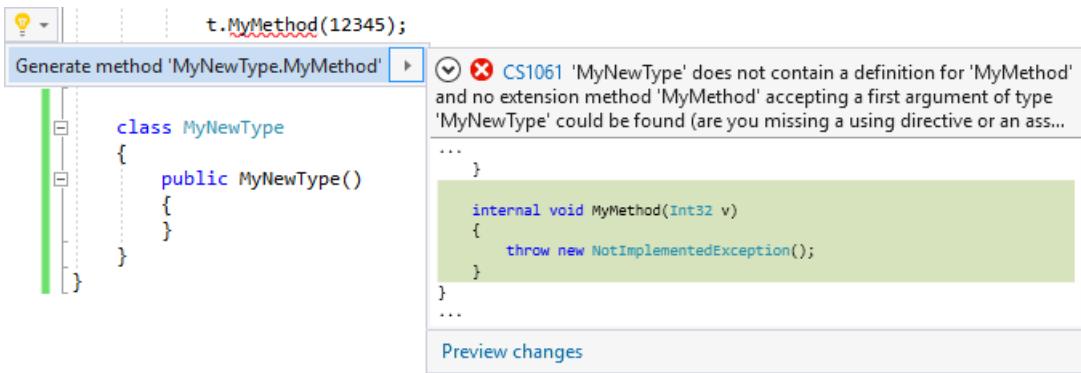
2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.

- **Mouse**

- Right-click and select the **Quick Actions and Refactorings** menu.
 - Hover over the red squiggle and click the icon which appears.
 - Click the icon which appears in the left margin if the text cursor is already on the line with the red squiggle.



3. Select **Generate method** from the drop-down menu.

TIP

Use the **Preview changes** link at the bottom of the preview window to see all of the changes that will be made before making your selection.

The method is created with any parameters inferred from its usage.

- C#:

```
internal void MyMethod(Int32 v)
{
    throw new NotImplementedException();
}
```

- Visual Basic:

```
Friend Sub MyMethod(v As Integer)
    Throw New NotImplementedException()
End Sub
```

See also

- [Code Generation](#)
- [Preview Changes](#)

Generate a field, property, or local variable in Visual Studio

2/28/2018 • 1 min to read • [Edit Online](#)

This code generation applies to:

- C#
- Visual Basic

What: Lets you immediately generate the code for a previously undeclared field, property, or local.

When: You introduce a new field, property or local while typing and want to properly declare it, automatically.

Why: You could declare the field, property or local before using it, however this feature will generate the declaration and type automatically.

How-to

1. Place your cursor on the line where there is a red squiggle. The red squiggle indicates a field, local or property that doesn't yet exist.

- C#:

```
static void Main(string[] args)
{
    double area = Math.PI * radius * radius;
```

- Visual Basic:

```
Sub Main()
    Dim area As Double = Math.PI * radius * radius
End Sub
```

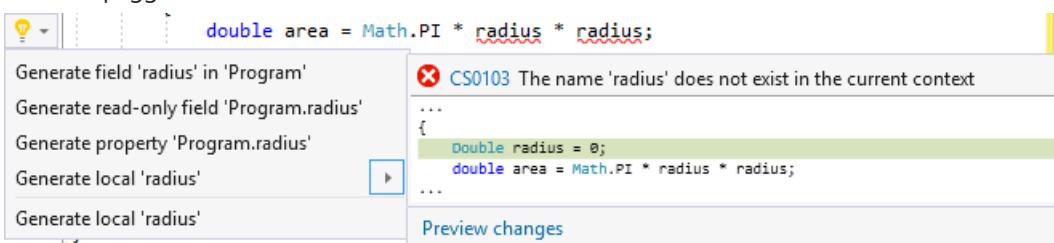
2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.

- **Mouse**

- Right-click and select the **Quick Actions and Refactorings** menu.
 - Hover over the red squiggle and click the icon which appears.
 - Click the icon which appears in the left margin if the text cursor is already on the line with the red squiggle.



3. Select one of the generation options from the drop-down menu.

TIP

Use the [Preview changes](#) link at the bottom of the preview window to see all of the changes that will be made before making your selection.

The field, property or local is created, with the type inferred from its usage.

- C#:

```
static void Main(string[] args)
{
    Int32 radius = 0;
    double area = Math.PI * radius * radius;
}
```

- Visual Basic:

```
Sub Main()
    Dim radius As Double = Nothing
    Dim area As Double = Math.PI * radius * radius
End Sub
```

See also

- [Code Generation](#)
- [Preview Changes](#)

Generate a constructor in Visual Studio

2/28/2018 • 3 min to read • [Edit Online](#)

This code generation applies to:

- C#
- Visual Basic

What: Lets you immediately generate the code for a new constructor on a class.

When: You introduce a new constructor and want to properly declare it automatically, or you modify an existing constructor.

Why: You could declare the constructor before using it, however this feature will generate it, with the proper parameters, automatically. Furthermore, modifying an existing constructor requires updating all the callsites unless you use this feature to update them automatically.

How: There are several ways to generate a constructor:

- [Generate constructor and pick members](#)
- [Generate constructor from selected fields](#)
- [Generate constructor from new usage](#)
- [Add parameter to existing constructor](#)
- [Create and initialize field/property from a constructor parameter](#)

Generate constructor and pick members (C# only)

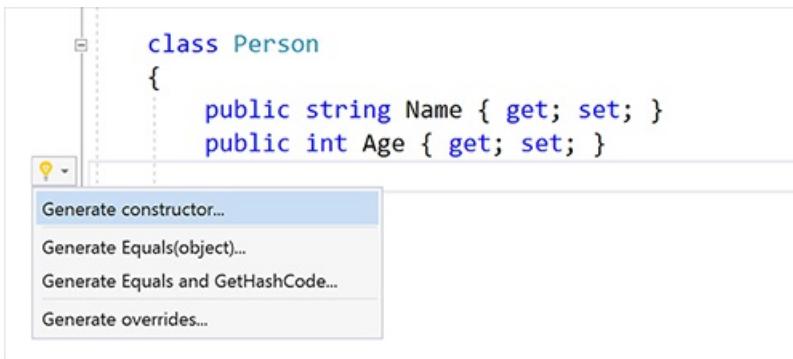
1. Place your cursor in any empty line in a class:

```
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

}
```

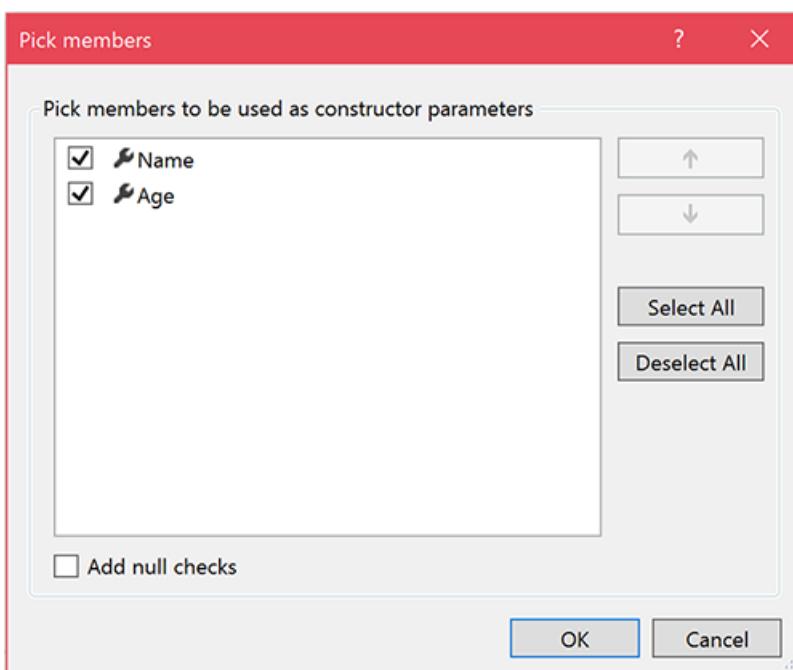
2. Next, do one of the following:

- **Keyboard**
 - Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.
- **Mouse**
 - Right-click and select the **Quick Actions and Refactorings** menu.
 - Click the  icon which appears in the left margin if the text cursor is already on the empty line in the class.



3. Select **Generate constructor...** from the drop-down menu.

- The **Pick members** dialog box opens.
4. Pick the members you want to include as constructor parameters. You can order them using the up and down arrows. Choose **OK**.



TIP

You can check the **Add null checks** checkbox to automatically generate null checks for your constructor parameters.

The constructor is created with the specified parameters.

```
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    public Person(string name, int age)
    {
        Name=name;
        Age=age;
    }
}
```

Generate constructor from selected fields (C# only)

1. Highlight the members you wish to have in your generated constructor:

```
class Person
{
    public string Name { get; set; }
    public string Surname { get; set; }
    public int Age { get; set; }
}
```

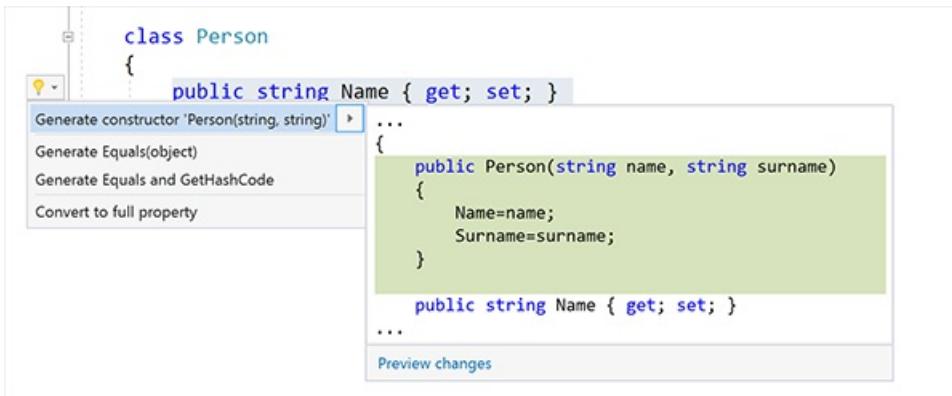
2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.

- **Mouse**

- Right-click and select the **Quick Actions and Refactorings** menu.
 - Click the  icon which appears in the left margin if the text cursor is already on the line with the selection.



3. Select **Generate constructor 'TypeName(...)'** from the drop-down menu.

The constructor is created with the selected parameters.

```
class Person
{
    public Person(string name, string surname)
    {
        Name=name;
        Surname=surname;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
    public int Age { get; set; }
}
```

Generate constructor from new usage (C# and Visual Basic)

1. Place your cursor on the line where there is a red squiggle. The red squiggle indicates a call to a constructor that doesn't yet exist.

- C#:

```

static void Main(string[] args)
{
    Person p = new Person("Bob", "Jones");
}

```

- Visual Basic:

```

Sub Main()
    Dim p As New Person("Bob", "Jones")
End Sub

```

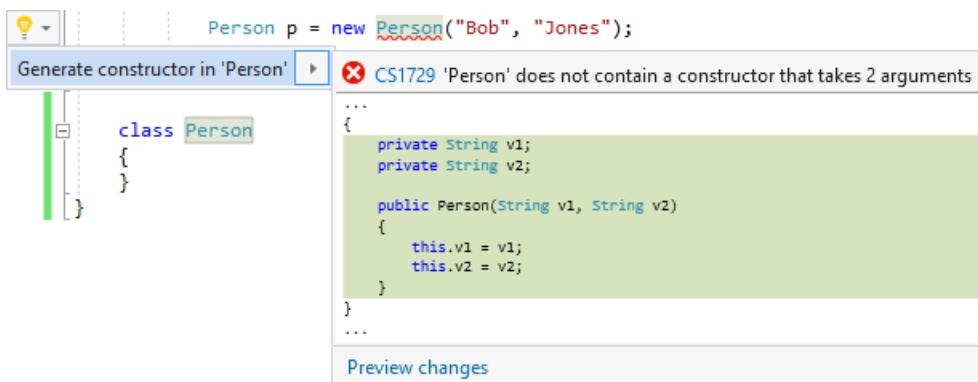
2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.

- **Mouse**

- Right-click and select the **Quick Actions and Refactorings** menu.
- Hover over the red squiggle and click the icon which appears.
- Click the icon which appears in the left margin if the text cursor is already on the line with the red squiggle.



3. Select **Generate constructor in 'TypeName'** from the drop-down menu.

TIP

Use the **Preview changes** link at the bottom of the preview window to see all of the changes that will be made before making your selection.

The constructor is created, with any parameters inferred from its usage.

- C#:

```

class Person
{
    private String v1;
    private String v2;

    public Person(String v1, String v2)
    {
        this.v1 = v1;
        this.v2 = v2;
    }
}

```

- Visual Basic:

```

Class Person
    Private v1 As String
    Private v2 As String

    Public Sub New(v1 As String, v2 As String)
        Me.v1 = v1
        Me.v2 = v2
    End Sub

    Public Property FirstName As String
    Public Property LastName As String
End Class

```

Add parameter to existing constructor (C# only)

1. Add a parameter to an existing constructor call.
2. Place your cursor on the line where there is a red squiggle indicating you've used a constructor that doesn't yet exist.

```

class Program
{
    static void Main(string[] args)
    {
        var p = new Person("John", "Smith", 30);
    }
}

class Person
{
    public Person(string name, string surname)
    {
        Name=name;
        Surname=surname;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
}

```

3. Next, do one of the following:

- **Keyboard**
 - Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.
- **Mouse**
 - Right-click and select the **Quick Actions and Refactorings** menu.
 - Hover over the red squiggle and click the  icon which appears.
 - Click the  icon which appears in the left margin if the text cursor is already on the line with the red squiggle.

```

class Program
{
    static void Main(string[] args)
    {
        var p = new Person("John", "Smith", 30);
    }
}

class Person
{
    public Person(string name, string surname)
    {
        Name=name;
        Surname=surname;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
}

```

4. Select **Add parameter to 'TypeName(...)'** from the drop-down menu.

The parameter is added to the constructor, with its type inferred from its usage.

```

class Program
{
    static void Main(string[] args)
    {
        var p = new Person("John", "Smith", 30);
    }
}

class Person
{
    public Person(string name, string surname, int v)
    {
        Name=name;
        Surname=surname;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
}

```

Create and initialize a field or property from a constructor parameter (C# only)

1. Find an existing constructor, and add a parameter:

```

class Person
{
    public Person(string name, string surname, int age)
    {
        Name=name;
        Surname=surname;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
}

```

2. Place your cursor inside the newly added parameter.

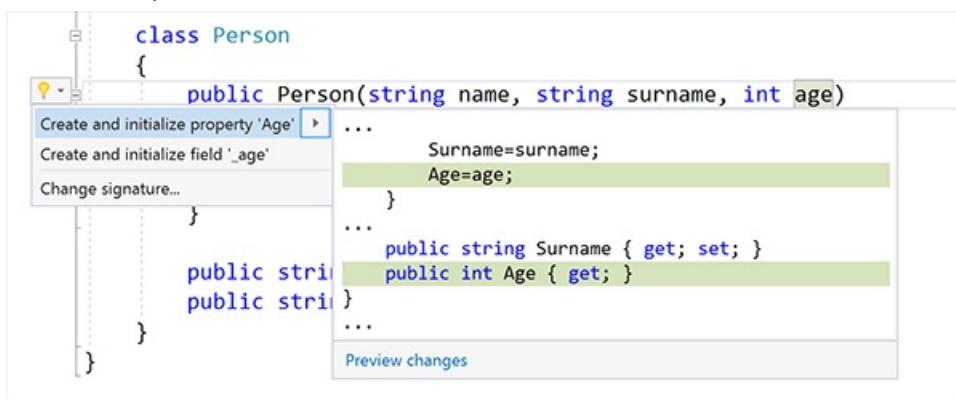
3. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.

- **Mouse**

- Right-click and select the **Quick Actions and Refactorings** menu.
- Click the  icon which appears in the left margin if the text cursor is already on the line with the added parameter.



4. Select **Create and initialize property** or **Create and initialize field** from the drop-down menu.

The field or property is declared and automatically named to match your types. A line of code is also added to initialize the field or property in the constructor body.

```
class Person
{
    public Person(string name, string surname, int age)
    {
        Name=name;
        Surname=surname;
        Age=age;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
    public int Age { get; }
}
```

See also

- [Code Generation](#)
- [Preview Changes](#)

Generate an override in Visual Studio

2/6/2018 • 1 min to read • [Edit Online](#)

This code generation applies to:

- C#
- Visual Basic

What: Lets you immediately generate the code for any method which can be overridden from a base class.

When: You want to override a base class method and generate the signature automatically.

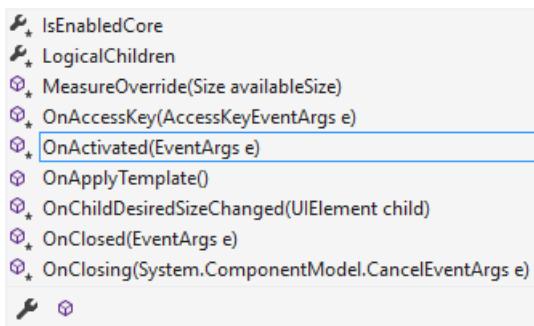
Why: You could write the method signature yourself, however this feature will generate the signature automatically.

How-to

1. Type `override` in C# or `Overrides` in Visual Basic, followed by a space, where you would like to insert an override method.

- C#:

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    override
} 
```

- Visual Basic:

```
Class MainWindow

    Overrides 
```

2. Select the method you want to override from the base class.

TIP

- Use the property icon  to show or hide properties in the list.
- Use the method icon  to show or hide methods in the list.

The selected method or property is added to the class as an override, ready to be implemented.

- C#:

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    protected override void OnActivated(EventArgs e)
    {
        base.OnActivated(e);
    }
}
```

- Visual Basic:

```
Class MainWindow

    Protected Overrides Sub OnActivated(e As EventArgs)
        MyBase.OnActivated(e)
    End Sub

End Class
```

See also

[Code Generation](#)

Generate Equals and GetHashCode method overrides in Visual Studio

2/28/2018 • 1 min to read • [Edit Online](#)

This code generation applies to:

- C#

What: Lets you generate **Equals** and **GetHashCode** methods.

When: Generate these overrides when you have a type that should be compared by one or more fields, instead of by object location in memory.

Why:

- If you are implementing a value type, you should consider overriding the **Equals** method to gain increased performance over the default implementation of the Equals method on ValueType.
- If you are implementing a reference type, you should consider overriding the **Equals** method if your type looks like a base type, such as Point, String, BigNumber, and so on.
- Override the **GetHashCode** method to allow a type to work correctly in a hash table. Read more guidance on [equality operators](#).

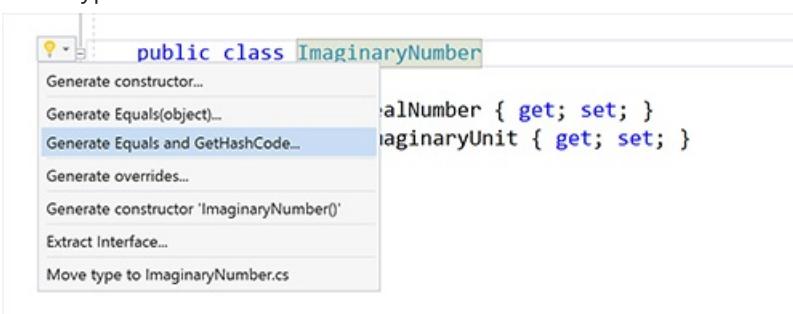
How-to

1. Place your cursor in your type declaration.

```
public class ImaginaryNumber
{
    public double RealNumber { get; set; }
    public double ImaginaryUnit { get; set; }
}
```

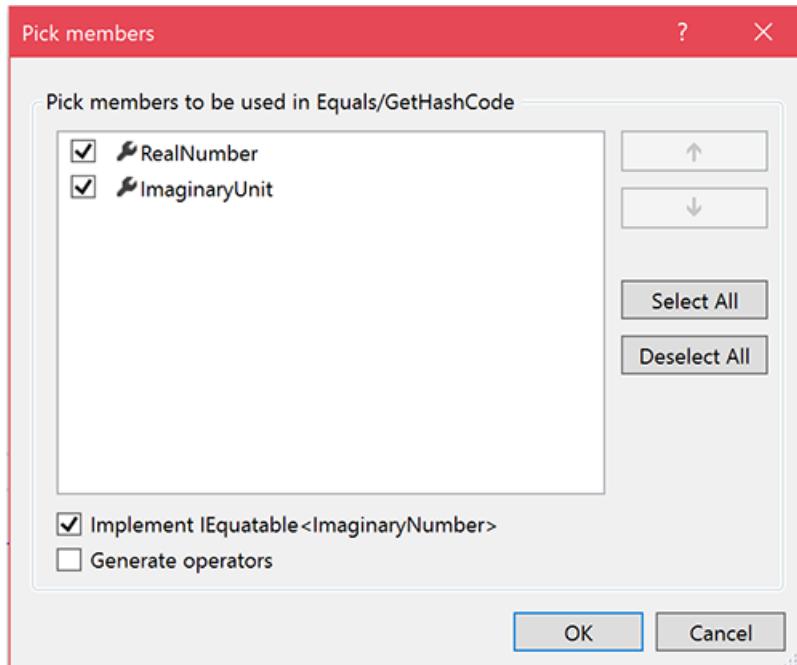
2. Next, do one of the following:

- **Keyboard**
 - Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.
- **Mouse**
 - Right-click and select the **Quick Actions and Refactorings** menu.
 - Click the  icon which appears in the left margin if the text cursor is already on the line with the type declaration.



3. Select **Generate Equals(object)** or **Generate Equals and GetHashCode** from the drop-down menu.

4. In the **Pick members** dialog box, select the members you want to generate the methods for:



TIP

You can also choose to generate operators from this dialog by using the checkboxes underneath the members list.

The Equals and GetHashCode overrides are generated with default implementations.

```
public class ImaginaryNumber : IEquatable<ImaginaryNumber>
{
    public double RealNumber { get; set; }
    public double ImaginaryUnit { get; set; }

    public override bool Equals(object obj)
    {
        return Equals(obj as ImaginaryNumber);
    }

    public bool Equals(ImaginaryNumber other)
    {
        return other != null && RealNumber == other.RealNumber
            && ImaginaryUnit == other.ImaginaryUnit;
    }

    public override int GetHashCode()
    {
        var hashCode = 352033288;
        hashCode = hashCode * -1521134295 + RealNumber.GetHashCode();
        hashCode = hashCode * -1521134295 + ImaginaryUnit.GetHashCode();
        return hashCode;
    }
}
```

See also

- [Code Generation](#)
- [Preview Changes](#)

Implement an abstract class in Visual Studio

2/28/2018 • 1 min to read • [Edit Online](#)

This code generation applies to:

- C#
- Visual Basic

What: Lets you immediately generate the code required to implement an abstract class.

When: You want to inherit from an abstract class.

Why: You could manually implement all abstract members one-by-one, however this feature will generate all method signatures automatically.

How-to

1. Place your cursor on the line where there is a red squiggle that indicates you have inherited from an abstract class, but have not implemented all required members.

- C#:

```
abstract class MyAbstractClass
{
    public abstract void Method1();
    public abstract int Method2(int value);
}

class MyClass : MyAbstractClass
{
}
```

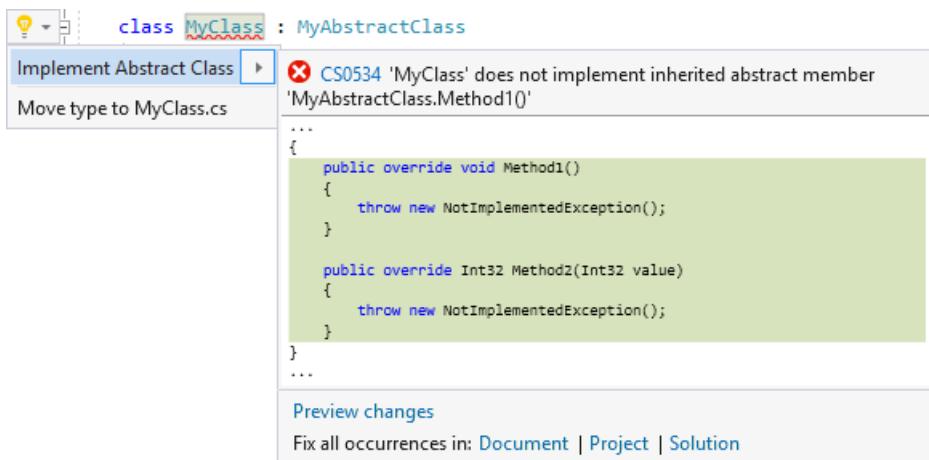
- Visual Basic:

```
Public MustInherit Class MyAbstractClass
    Public MustOverride Sub Method1()
    Public MustOverride Function Method2(value As Integer) As Integer
End Class

Public Class MySubClass
    Inherits MyAbstractClass
End Class
```

2. Next, do one of the following:

- **Keyboard**
 - Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.
- **Mouse**
 - Right-click and select the **Quick Actions and Refactorings** menu.
 - Hover over the red squiggle and click the  icon which appears.
 - Click the  icon which appears in the left margin if the text cursor is already on the line with the red squiggle.



3. Select **Implement Abstract Class** from the drop-down menu.

TIP

- Use the **Preview changes** link at the bottom of the preview window to see all of the changes that will be made before making your selection.
- Use the **Document**, **Project**, and **Solution** links at the bottom of the preview window to create the proper method signatures across multiple classes that inherit from the abstract class.

The abstract method signatures are created, and are ready to be implemented.

- C#:

```
class MyClass : MyAbstractClass
{
    public override void Method1()
    {
        throw new NotImplementedException();
    }

    public override Int32 Method2(Int32 value)
    {
        throw new NotImplementedException();
    }
}
```

- Visual Basic:

```
Public Class MySubClass
    Inherits MyAbstractClass
    Public Overrides Sub Method1()
        Throw New NotImplementedException()
    End Sub

    Public Overrides Function Method2(value As Integer) As Integer
        Throw New NotImplementedException()
    End Function
End Class

End Module
```

See also

- [Code Generation](#)
- [Preview Changes](#)

Implement an interface in Visual Studio

2/28/2018 • 1 min to read • [Edit Online](#)

This code generation applies to:

- C#
- Visual Basic

What: Lets you immediately generate the code required to implement an interface.

When: You want to inherit from an interface.

Why: You could manually implement all interface one-by-one, however this feature will generate all method signatures automatically.

How-to

1. Place your cursor on the line where there is a red squiggle that indicates you have referenced an interface, but have not implemented all required members.

- C#:

```
interface IMyInterface
{
    void Method1();
    int Method2(int value);
}

class MyClass : IMyInterface
{
}
```

- Visual Basic:

```
Interface IMyInterface
    Sub Method1()
        Function Method2(value As Integer) As Integer
    End Interface

    Class MyImplementation
        Implements IMyInterface
    End Class
```

2. Next, do one of the following:

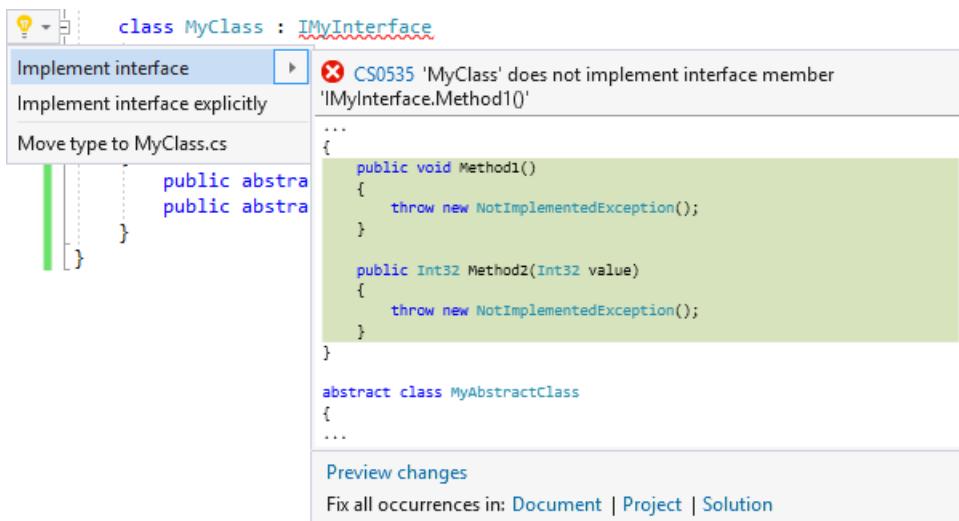
- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.

- **Mouse**

- Right-click and select the **Quick Actions and Refactorings** menu.
 - Hover over the red squiggle and click the  icon which appears.
 - Click the  icon which appears in the left margin if the text cursor is already on the line with the red squiggle.

3. Select **Implement interface** from the drop-down menu.



TIP

- Use the **Preview changes** link at the bottom of the preview window to see all of the changes that will be made before making your selection.
- Use the **Document**, **Project**, and **Solution** links at the bottom of the preview window to create the proper method signatures across multiple classes that implement the interface.

The interface's method signatures is created, and is ready to be implemented.

- C#:

```
interface IMyInterface
{
    void Method1();
    int Method2(int value);
}

class MyClass : IMyInterface
{
    public void Method1()
    {
        throw new NotImplementedException();
    }

    public Int32 Method2(Int32 value)
    {
        throw new NotImplementedException();
    }
}
```

- Visual Basic:

```
Class MyImplementation
    Implements IMyInterface

    Public Sub Method1() Implements IMyInterface.Method1
        Throw New NotImplementedException()
    End Sub

    Public Function Method2(value As Integer) As Integer Implements IMyInterface.Method2
        Throw New NotImplementedException()
    End Function
End Class
```

TIP

(C# only) Use the **Implement interface explicitly** option to preface each generated method with the interface name to avoid name collisions.

```
interface IMyInterface
{
    void Method1();
    int Method2(int value);
}

class MyClass : IMyInterface
{
    void IMyInterface.Method1()
    {
        throw new NotImplementedException();
    }

    Int32 IMyInterface.Method2(Int32 value)
    {
        throw new NotImplementedException();
    }
}
```

See also

- [Code Generation](#)
- [Preview Changes](#)

Introduce a local variable in Visual Studio

2/28/2018 • 1 min to read • [Edit Online](#)

This code generation applies to:

- C#
- Visual Basic

What: Lets you immediately generate a local variable to replace an existing expression.

When: You have code which could be easily reused later if it were in a local variable.

Why: You could copy and paste the code multiple times to use it in various locations, however it would be better to perform the operation once, store the result in a local variable, and use the local variable throughout.

How-to

1. Highlight the expression that you want to assign to a new local variable.

- C#:

```
static void Main(string[] args)
{
    Debug.WriteLine(new Random().Next());
}
```

- Visual Basic:

```
Sub Main()
    Debug.WriteLine(New Random().Next())
End Sub
```

2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.

- **Mouse**

- Right-click and select the **Quick Actions and Refactorings** menu.
 - Click the icon which appears in the left margin if the text cursor is already on the line with the red squiggle.



3. Select **Introduce local for (all occurrences of) 'expression'** from the drop-down menu.

TIP

Use the **Preview changes** link at the bottom of the preview window to see all of the changes that will be made before making your selection.

The local variable is created, with the type inferred from its usage. Give the new local variable a new name.

- C#:

```
static void Main(string[] args)
{
    Random random = new Random();
    Debug.WriteLine(random.Next());
}
```

- Visual Basic:

```
Sub Main()
    Dim random As Random = New Random()
    Debug.WriteLine(random.Next())
End Sub
```

NOTE

You can use the **...all occurrences of...** menu option to replace every instance of the selected expression, not just the one you have specifically highlighted.

See also

- [Code Generation](#)
- [Preview Changes](#)

Refactoring code

1/13/2018 • 1 min to read • [Edit Online](#)

Refactoring is the process of modifying code in order to make it easier to maintain, understand, and extend, but without changing its behavior.

Different refactoring operations are available for different programming languages in Visual Studio. The topics in this section cover the refactorings available for C# and Visual Basic. For information about refactoring C++, see [Writing and refactoring code \(C++\)](#). Refactoring support for F# is provided by the [Visual F# Power Tools](#), a third-party Visual Studio extension.

See also

- [Quick Actions](#)
- [Visual Studio IDE](#)
- [Writing Code in the Code and Text Editor](#)
- [Preview Changes](#)

Change a method signature refactoring

2/28/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#
- Visual Basic

What: Lets you remove or change the order of a method's parameters.

When: You want to move or remove a method parameter that is currently being used in a variety of locations.

Why: You could manually remove and re-order the parameters, and then find all calls to that method and change them one-by-one, but that could lead to errors. This refactoring tool will perform the task automatically.

How-to

1. Highlight or place the text cursor inside the name of the method to modify, or one of its usages:

- C#:

```
static void Main(string[] args)
{
    ChangeName("John", "Doe");
}

1 reference
private static void ChangeName(string firstName, string lastName)
```

- VB:

```
Sub Main()
    ChangeName("John", "Doe")
End Sub

Sub ChangeName(firstName As String, lastName As String)
End Sub
```

2. Next, do one of the following:

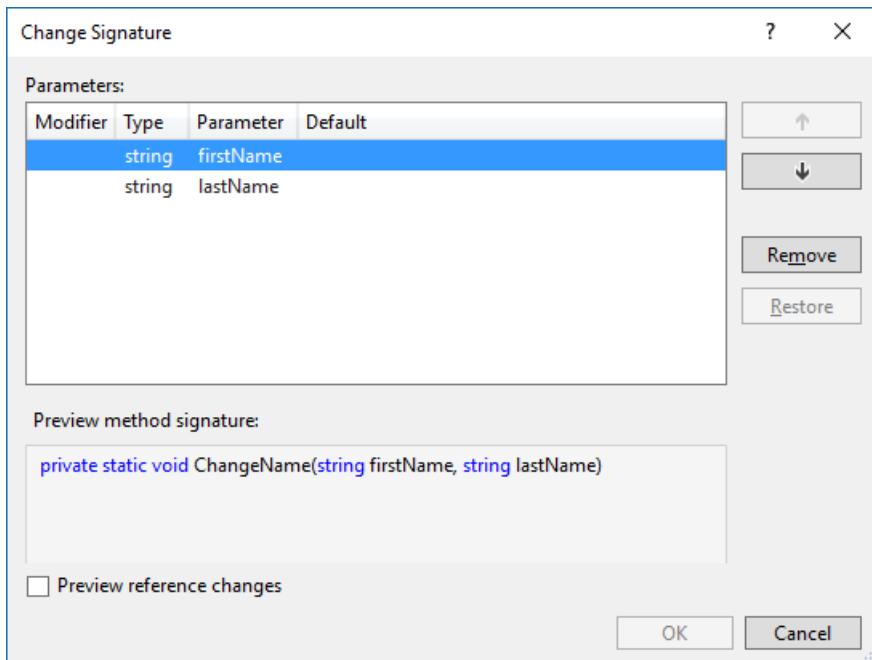
- **Keyboard**

- Press **Ctrl+R**, then **Ctrl+V**. (Note that your keyboard shortcut may be different based on which profile you've selected.)
- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Change Signature** from the Preview window popup.

- **Mouse**

- Select **Edit > Refactor > Remove Parameters**.
- Select **Edit > Refactor > Reorder Parameters**.
- Right-click the code, select the **Quick Actions and Refactorings** menu and select **Change Signature** from the Preview window popup.

3. In the **Change Signature** dialog that pops up, you can use the buttons on the right side to change the method signature:



| BUTTON | DESCRIPTION |
|----------------|---|
| Up/Down | Move the selected parameter up and down the list |
| Remove | Remove the selected parameter from the list |
| Restore | Restore the selected, crossed-out parameter to the list |

TIP

Use the **Preview reference changes** checkbox to see what the result will be before committing to it.

- When you are finished, press the **OK** button to make the changes.

- C#:

```
static void Main(string[] args)
{
    ChangeName("Doe", "John");
}

1 reference
private static void ChangeName(string lastName, string firstName)
{}
```

- Visual Basic:

```
Sub Main()
    ChangeName("Doe", "John")
End Sub

Sub ChangeName(lastName As String, firstName As String)
End Sub
```

See also

- [Refactoring](#)
- [Preview Changes](#)

Convert Get method to property / Convert property to Get method refactorings

2/28/2018 • 1 min to read • [Edit Online](#)

These refactorings apply to:

- C#

Convert Get method to property

What: Lets you convert a Get method into a property (and optionally your Set method), and vice versa.

When: You have a Get method that does not contain any logic.

How-to

1. Place your cursor in your Get method name.
2. Next, do one of the following:
 - **Keyboard**
 - Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu, and select **Replace method with property** from the Preview window popup.
 - **Mouse**
 - Right-click the code, select the **Quick Actions and Refactorings** menu, and select **Replace method with property** from the Preview window popup.
3. (Optional) If you have a Set method, you can also convert your Set method at this time by selecting **Replace Get method and Set method with property**.
4. If you are happy with the change in the code preview, press **Enter** or click the fix from the menu and the changes will be committed.

Example:

```
private int MyValue;

// Before
public int GetMyValue()
{
    return MyValue;
}

// Replace 'GetMyValue' with property

// After
public int MyValue
{
    get { return MyValue; }
}
```

Convert property to Get method

What: Lets you convert a property to a Get method

When: You have a property that involves more than immediately setting and getting a value

How-to

1. Place your cursor in your Get method name.

2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Replace property with methods** from the Preview window popup.

- **Mouse**

- Right-click the code, select the **Quick Actions and Refactorings** menu and select **Replace property with methods** from the Preview window popup.

3. If you are happy with the change in the code preview, press **Enter** or click the fix from the menu and the changes will be committed.

See also

- [Refactoring](#)
- [Preview Changes](#)

Encapsulate a field refactoring

2/28/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#
- Visual Basic

What: Lets you turn a field into a property, and update all usages of that field to use the newly created property.

When: You want to move a field into a property, and update all references to that field.

Why: You want to give other classes access to a field, but don't want those classes to have direct access. By wrapping the field in a property, you could write code to verify the value being assigned, for example.

How-to

1. Highlight or place the text cursor inside the name of the field to encapsulate:

- C#:

```
class Square
{
    public double side;
}

class Program
{
    static void Main(string[] args)
    {
        Square s = new Square();
        s.side = 1.23;
    }
}
```

- Visual Basic:

```
Class Square
    Public side As Double
End Class

Module Module1
    Sub Main()
        Dim s As New Square
        s.side = 1.23
    End Sub
End Module
```

2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+R**, then **Ctrl+E**. (Note that your keyboard shortcut may be different based on which profile you've selected.)
- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select either **Encapsulate field** entry from the Preview window popup.

- **Mouse**

- Select **Edit > Refactor > Encapsulate Field**.
- Right-click the code, select the **Quick Actions and Refactorings** menu and select either **Encapsulate field** entry from the Preview window popup.

| SELECTION | DESCRIPTION |
|--|---|
| Encapsulate field (and use property) | Encapsulates the field with a property, and updates all usages of the field to use the generated property |
| Encapsulate field (but still use field) | Encapsulates the field with a property, but leaves all usages of the field untouched |

The property is created and references to the field are updated, if selected.

TIP

Use the [Preview changes](#) link in the popup window to see what the result will be before committing to it.

- C#:

```
class Square
{
    private double side;

    public double Side
    {
        get
        {
            return side;
        }

        set
        {
            side = value;
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        Square s = new Square();
        s.Side = 1.23;
    }
}
```

- Visual Basic:

```
Class Square
    Private _side As Double

    Public Property Side As Double
        Get
            Return _side
        End Get
        Set(value As Double)
            _side = value
        End Set
    End Property
End Class

Module Module1
    Sub Main()
        Dim s As New Square
        s.Side = 1.23
    End Sub
End Module
```

See also

- [Refactoring](#)

- Preview Changes

Extract an interface refactoring

2/28/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#
- Visual Basic

What: Lets you create an interface using existing members from a class, struct or interface.

When: You have several classes, structs or interfaces with methods that could be made common and used by other classes, structs or interfaces.

Why: Interfaces are great constructs for object-oriented designs. Imagine having classes for various animals (Dog, Cat, Bird) which might all have common methods, such as Eat, Drink, Sleep. Using an interface like IAnimal would allow Dog, Cat, and Bird to have a common "signature" for these methods.

How-to

1. Highlight the name of the class to perform the action on, or just put the text cursor somewhere in the class name.

- C#:

```
class Dog
{
    public void Eat()
    {
    }

    public void Drink(int value)
    {
    }

    public int Sleep()
    {
        return 0;
    }
}
```

- Visual Basic:

```
Class Dog
    Public Sub Eat()

    End Sub

    Public Sub Drink(value As Integer)

    End Sub

    Public Function Sleep() As Integer
        Return 0
    End Function
End Class
```

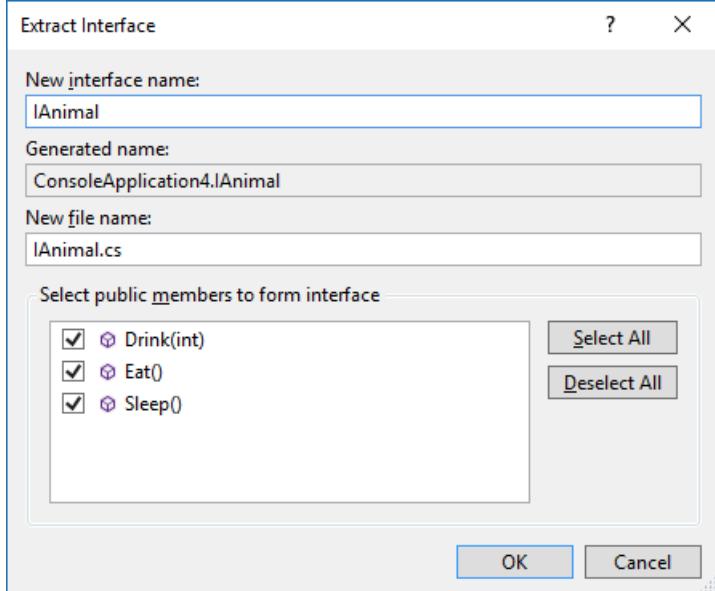
2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+R**, then **Ctrl+I**. (Note that your keyboard shortcut may be different based on which profile you've selected.)

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Extract Interface** from the Preview window popup.
- **Mouse**
 - Select **Edit > Refactor > Extract Interface**.
 - Right-click the name of the class, select the **Quick Actions and Refactorings** menu and select **Extract Interface** from the Preview window popup.

3. In the **Extract Interface** dialog box that pops up, enter the information asked:



| FIELD | DESCRIPTION |
|--|--|
| New interface name | The name of the interface to be created. This will default to <code>IClassName</code> , where <i>ClassName</i> is the name of the class you selected above. |
| New file name | The name of the file which will be generated that will contain the interface. As with the interface name, this will default to <code>IClassName</code> , where <i>ClassName</i> is the name of the class you selected above. |
| Select public members to form interface | The items to extract into the interface. You may select as many as you wish. |

4. Choose **OK**.

The interface is created in the file of the name specified. Additionally, the class you selected implements that interface.

- C#:

```

class Dog : IAnimal
{
    public void Eat()
    {
    }

    public void Drink(int value)
    {
    }

    public int Sleep()
    {
        return 0;
    }
}

interface IAnimal
{
    void Drink(int value);
    void Eat();
    int Sleep();
}

```

- Visual Basic:

```

Class Dog
    Implements IAnimal

    Public Sub Eat() Implements IAnimal.Eat
    End Sub

    Public Sub Drink(value As Integer) Implements IAnimal.Drink
    End Sub

    Public Function Sleep() As Integer Implements IAnimal.Sleep
        Return 0
    End Function
End Class

Interface IAnimal
    Sub Drink(value As Integer)
    Sub Eat()
    Function Sleep() As Integer
End Interface

```

See also

[Refactoring](#)

Extract a method refactoring

2/28/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#
- Visual Basic

What: Lets you turn a fragment of code into its own method.

When: You have a fragment of existing code in some method that needs to be called from another method.

Why: You could copy/paste that code, but that would lead to duplication. A better solution is to refactor that fragment into its own method which can be called freely by any other method.

How-to

1. Highlight the code to be extracted:

- C#:

```
class Program
{
    static void Main(string[] args)
    {
        double radius = 1.23;

        double area = Math.PI * radius * radius;
    }
}
```

- Visual Basic:

```
Sub Main()
    Dim radius As Double

    Dim area As Double = Math.PI * radius * radius
End Sub
```

2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+R**, then **Ctrl+M**. (Note that your keyboard shortcut may be different based on which profile you've selected.)
- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Extract Method** from the Preview window popup.

- **Mouse**

- Select **Edit > Refactor > Extract Method**.
- Right-click the code and select **Refactor > Extract > Extract Method**.
- Right-click the code, select the **Quick Actions and Refactorings** menu and select **Extract Method** from the Preview window popup.

The method will be immediately created. From here, you can now rename the method simply by typing the new name.

TIP

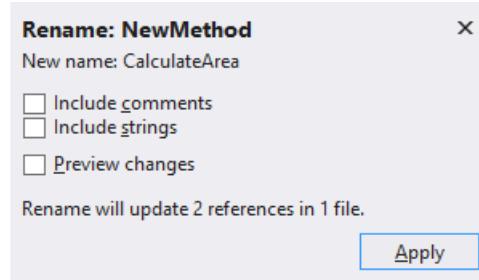
You can also update comments and other strings to use this new name, as well as [preview changes](#) before saving, using the checkboxes in the **Rename** box which appears at the top right of your IDE.

- C#:

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double radius = 1.23;
            CalculateArea(radius);
        }

        private static void CalculateArea(Double radius)
        {
            double area = Math.PI * radius * radius;
        }
    }
}
```

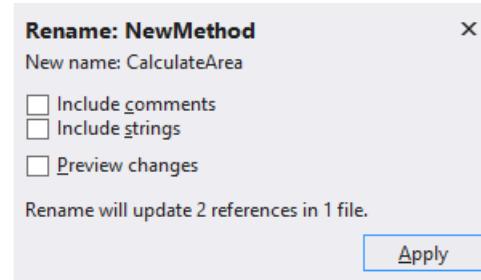


- Visual Basic:

```
Module Module1
    Sub Main()
        Dim radius As Double

        CalculateArea(radius)
    End Sub

    Private Sub CalculateArea(radius As Double)
        Dim area As Double = Math.PI * radius * radius
    End Sub
End Module
```



3. When you're happy with the change, choose the **Apply** button or press **Enter** and the changes will be committed.

See also

- [Refactoring](#)
- [Preview Changes](#)

Inline a temporary variable refactoring

2/28/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#
- Visual Basic

What: Lets you remove a temporary variable and replace it with its value instead.

When: The use of the temporary variable makes the code harder to understand.

Why: Removing a temporary variable may make the code easier to read.

How-to

1. Highlight or place the text cursor inside the temporary variable to be inlined:

- C#:

```
static void Main(string[] args)
{
    double radiusSquared = 1.23 * 1.23;
    double area = Math.PI * radiusSquared;
}
```

- Visual Basic:

```
Dim radiusSquared As Double = 1.23 * 1.23
Dim area = Math.PI * radiusSquared
```

2. Next, do one of the following:

- **Keyboard**
 - Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu.
- **Mouse**
 - Right-click the code and select the **Quick Actions and Refactorings** menu.

3. Select **Inline temporary variable** from the Preview window popup.

The variable is removed and its usages replaced by the value of the variable.

- C#:

```
static void Main(string[] args)
{
    double area = Math.PI * 1.23 * 1.23;
}
```

- Visual Basic:

```
Dim area = Math.PI * 1.23 * 1.23
```

See also

[Refactoring](#)

Move declaration near reference refactoring

2/28/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#

What: Lets you move variable declarations closer to their usage.

When: You have variable declarations that can be in a narrower scope.

Why: You could leave it as it is, but that may cause readability issues or information hiding. This is a chance to refactor to improve readability.

How-to

1. Place your cursor in the variable declaration.

2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Move declaration near reference** from the Preview window popup.

- **Mouse**

- Right-click the code, select the **Quick Actions and Refactorings** menu and select **Move declaration near reference** from the Preview window popup.

3. When you're happy with the change, press **Enter** or click the fix in the menu and the changes will be committed.

Example:

```
// Before
int x;
if (condition)
{
    x = 1;
    Console.WriteLine(x);
}

// Move declaration near reference

// After
if (condition)
{
    int x = 1;
    Console.WriteLine(x);
}
```

See also

- [Refactoring](#)
- [Preview Changes](#)

Move a type to a matching file refactoring

2/28/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#
- Visual Basic

What: Lets you move the selected type to a separate file with the same name.

When: You have multiple classes, structs, interfaces, etc. in the same file which you want to separate.

Why: Placing multiple types in the same file can make it difficult to find these types. By moving types to files with the same name, code becomes more readable and easier to navigate.

How-to

1. Highlight or place the text cursor inside the name of the type to move:

- C#:

```
class Person
{
    // References
    public string FirstName { get; set; }
    // References
    public string LastName { get; set; }
}
```

- Visual Basic:

```
Class Person
    Public Property FirstName As String
    Public Property LastName As String
End Class
```

2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Move type to TypeName.cs** from the Preview window popup, where *TypeName* is the name of the type you have selected.

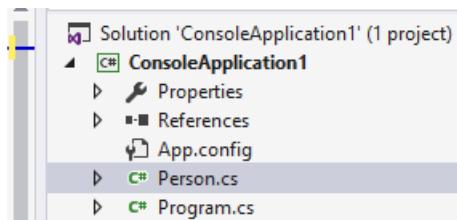
- **Mouse**

- Right-click the code, select the **Quick Actions and Refactorings** menu and select **Move type to TypeName.cs** from the Preview window popup, where *TypeName* is the name of the type you have selected.

The type is moved to a new file with that name, as part of your solution.

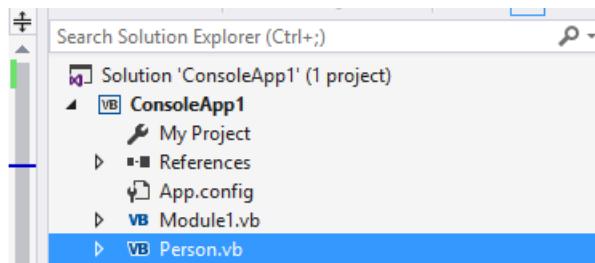
- C#:

```
0 references
class Person
{
    // References
    public string FirstName { get; set; }
    // References
    public string LastName { get; set; }
}
```



- Visual Basic:

```
Class Person
    Public Property FirstName As String
    Public Property LastName As String
End Class
```



See also

[Refactoring](#)

Remove unreachable code refactoring

2/28/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#

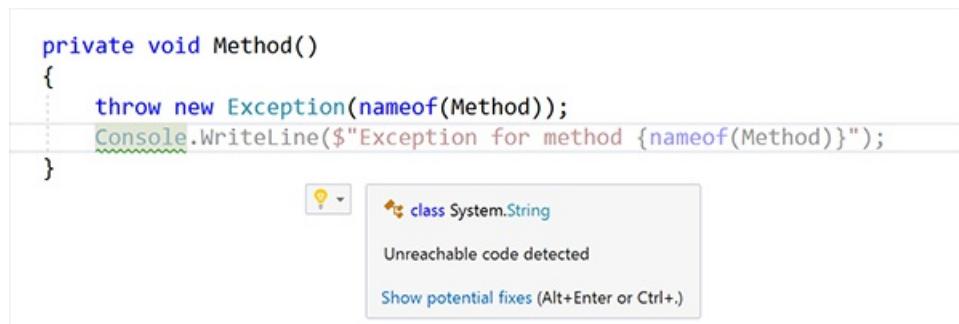
What: Removes code that will never be executed.

When: Your program has no path to a code snippet, making that code snippet unnecessary.

Why: Improve readability and maintainability by removing code that is superfluous and will never be executed.

How-to

1. Place your cursor anywhere in the faded out code that is unreachable:



1. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Remove unreachable code** from the Preview window popup.

- **Mouse**

- Right-click the code, select the **Quick Actions and Refactorings** menu and select **Remove unreachable code** from the Preview window popup.

2. When you're happy with the change, press **Enter** or click the fix in the menu and the changes will be committed.

Example:

```
// Before
private void Method()
{
    throw new Exception(nameof(Method));
    Console.WriteLine($"Exception for method {nameof(Method)}");
}

// Remove unreachable code

// After
private void Method()
{
    throw new Exception(nameof(Method));
}
```

See also

- [Refactoring](#)
- [Preview Changes](#)

Rename a code symbol refactoring

2/6/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#
- Visual Basic

What: Lets you rename identifiers for code symbols, such as fields, local variables, methods, namespaces, properties and types.

When: You want to safely rename something without having to find all instances, and copy/paste the new name.

Why: Copy and pasting the new name across an entire project would likely result in errors. This refactoring tool will accurately perform the renaming action.

How-to

1. Highlight or place the text cursor inside the item to be renamed:

- C#:

```
static void Main(string[] args)
{
    double r = 1.23;
    double area = Math.PI * r * r;
}
```

- Visual Basic:

```
Dim r As Double = 1.23
Dim area As Double = Math.PI * r * r
```

2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+R**, then **Ctrl+R**. (Note that your keyboard shortcut may be different based on which profile you've selected.)

- **Mouse**

- Select **Edit > Refactor > Rename**.
 - Right-click the code and select **Rename**.

3. Rename the item simply by typing the new name.

- C#:

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double r = 1.23;

            double area = Math.PI * r * r;
        }
    }
}
```

- Visual Basic:

```
Dim radius As Double = 1.23

Dim area As Double = Math.PI * radius * radius
```

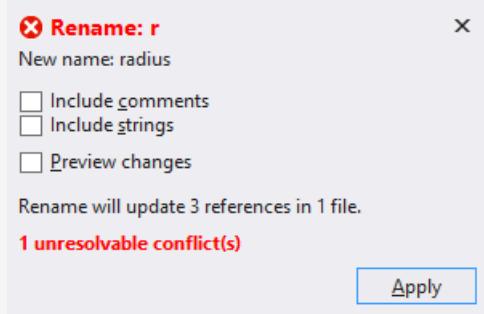
TIP

You can also update comments and other strings to use this new name, as well as [preview the changes](#) before saving, using the checkboxes in the **Rename** box which appears at the top right of your editor.

4. When you're happy with the change, choose the **Apply** button or press **Enter** and the changes will be committed.

NOTE

If you use a name that already exists which would cause a conflict, the **Rename** box will warn you.



See also

[Refactoring](#)

[Preview Changes](#)

Sync a type to a filename, or a filename to a type refactoring

2/28/2018 • 1 min to read • [Edit Online](#)

This refactoring applies to:

- C#
- Visual Basic

What: Lets you rename a type to match the filename, or rename a filename to match the type it contains.

When: You have renamed a file or type and haven't yet updated the corresponding file or type to match.

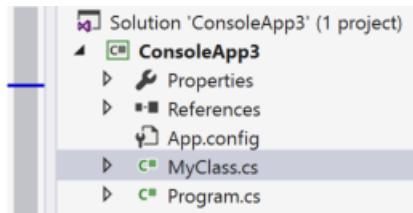
Why: Placing a type in a file with a different name, or vice-versa, it difficult to find what you're looking for. By renaming either the type or filename, code becomes more readable and easier to navigate.

How-to

1. Highlight or place the text cursor inside the name of the type to synchronize:

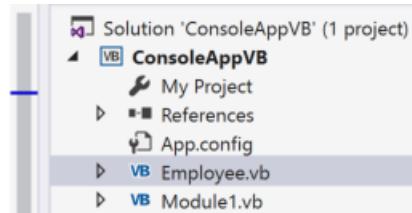
- C#:

```
public class MyNewClass
{
    public void Method() { }
```



- Visual Basic:

```
Class Person
    Public Sub Method()
        End Sub
    End Class
```



2. Next, do one of the following:

- **Keyboard**

- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Rename file to TypeName.cs** from the Preview window popup, where *TypeName* is the name of the type you have selected.
- Press **Ctrl+.** to trigger the **Quick Actions and Refactorings** menu and select **Rename type to Filename** from the Preview window popup, where *Filename* is the name of the current file.

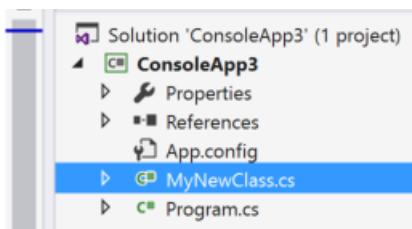
- **Mouse**

- Right-click the code, select the **Quick Actions and Refactorings** menu, and select **Rename file to TypeName.cs** from the Preview window popup, where *TypeName* is the name of the type you have selected.
- Right-click the code, select the **Quick Actions and Refactorings** menu, and select **Rename type to Filename** from the Preview window popup, where *Filename* is the name of the current file.

The type or file is renamed.

- C#: In the example below, the file **MyClass.cs** was renamed to **MyNewClass.cs** to match the type name.

```
public class MyNewClass
{
    public void Method() { }
```



- Visual Basic: In the example below, the file **Employee.vb** was renamed to **Person.vb** to match the type name.

```
Class Person
    Public Sub Method()
        End Sub
    End Class
```



![NOTE] This refactoring is not yet available for .NET Standard and .NET Core projects.

See also

[Refactoring](#)

Walkthrough: Test-first development with the Generate From Usage feature

1/13/2018 • 7 min to read • [Edit Online](#)

This topic demonstrates how to use the [Generate From Usage](#) feature, which supports test-first development.

Test-first development is an approach to software design in which you first write unit tests based on product specifications, and then write the source code that is required to make the tests succeed. Visual Studio supports test-first development by generating new types and members in the source code when you first reference them in your test cases, before they are defined.

Visual Studio generates the new types and members with minimal interruption to your workflow. You can create stubs for types, methods, properties, fields, or constructors without leaving your current location in code. When you open a dialog box to specify options for type generation, the focus returns immediately to the current open file when the dialog box closes.

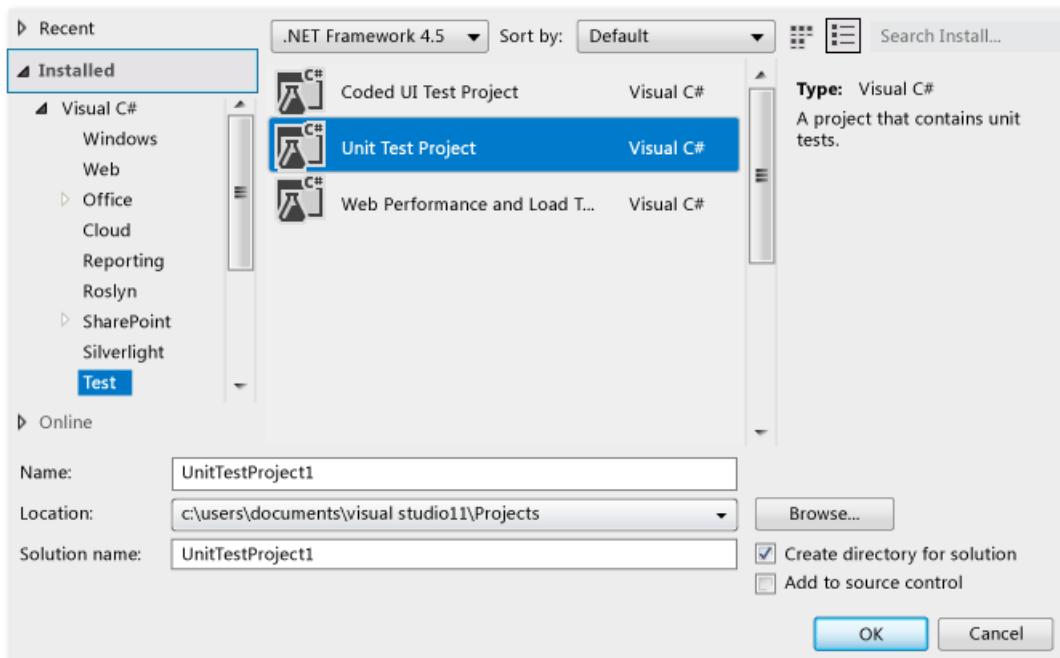
The Generate From Usage feature can be used with test frameworks that integrate with Visual Studio. In this topic, the Microsoft Unit Testing Framework is demonstrated.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

To create a Windows Class Library project and a Test project

1. In Visual C# or Visual Basic, create a new **Windows Class Library** project. Name it `GFUDemo_VB` or `GFUDemo_CS`, depending on which language you are using.
2. In **Solution Explorer**, right-click the solution icon at the top, choose **Add**, and then choose **New Project**. In the left pane of the **New Project** dialog box, choose **Test**.
3. In the middle pane, choose **Unit Test Project** and accept the default name of **UnitTestProject1**. The following illustration shows the dialog box when it appears in Visual C#. In Visual Basic, the dialog box looks similar.



- Choose **OK** to close the **New Project** dialog box.

To add a reference to the Class Library project

- In **Solution Explorer**, under your unit test project, right-click the **References** entry and choose **Add Reference**.
- In the **Reference Manager** dialog box, select **Projects** and then select the class library project.
- Choose **OK** to close the **Reference Manager** dialog box.
- Save your solution. You are now ready to begin writing tests.

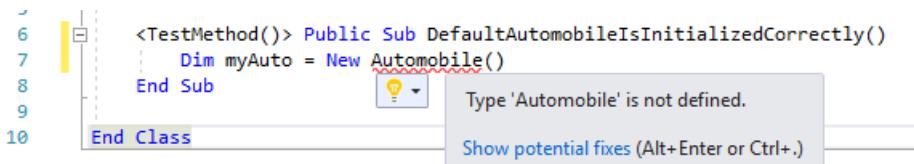
To generate a new class from a unit test

- The test project contains a file that is named `UnitTest1`. Double-click this file in **Solution Explorer** to open it in the code editor. A test class and test method have been generated.
- Locate the declaration for class `UnitTest1` and rename it to `AutomobileTest`.

NOTE

IntelliSense now provides two alternatives for IntelliSense statement completion: *completion mode* and *suggestion mode*. Use suggestion mode for situations in which classes and members are used before they are defined. When an IntelliSense window is open, you can press **Ctrl+Alt+SPACEBAR** to toggle between completion mode and suggestion mode. See [Using IntelliSense](#) for more information. Suggestion mode will help when you are typing `Automobile` in the next step.

- Locate the `TestMethod1()` method and rename it to `DefaultAutomobileIsInitializedCorrectly()`. Inside this method, create a new instance of a class named `Automobile`, as shown in the following screenshots. A wavy underline appears, which indicates a compile-time error, and a **Quick Actions** light bulb appears in the left margin (C# only), or directly below the squiggle if you hover over it.

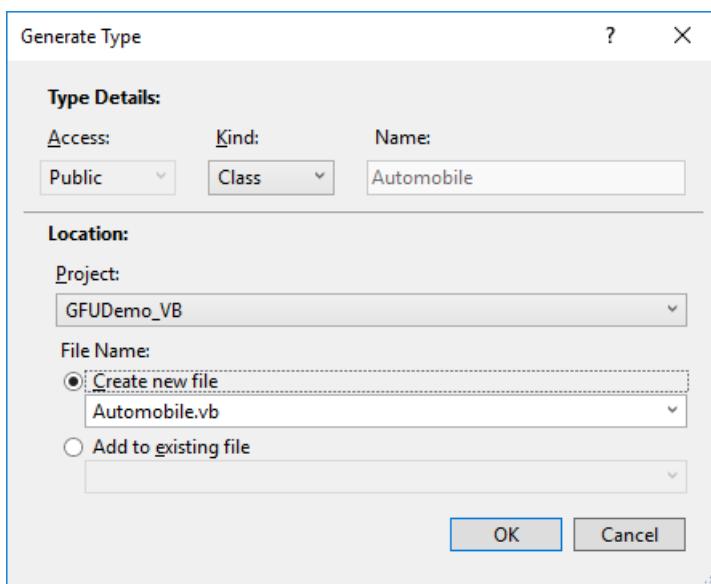


```

10
11
12
13 [TestMethod]
14     public void DefaultAutomobileIsInitializedCorrectly()
15     {
16         Automobile myAuto = new Automobile();

```

4. Choose or click the Quick Actions light bulb. You'll see an error message that states that the type `Automobile` is not defined. You are also presented with some solutions.
5. Click **Generate new type...** to open the **Generate Type** dialog box. This dialog box provides options that include generating the type in a different project.
6. In the **Project** list, click **GFUDemo_VB** or **GFUDemo_CS** to instruct Visual Studio to add the file to the class library project instead of the test project. If it's not already selected, choose **Create new file** and name it **Automobile.cs** or **Automobile.vb**.



7. Click **OK** to close the dialog box and create the new file.
8. In **Solution Explorer**, look under the GFUDemo_VB or GFUDemo_CS project node to verify that the new `Automobile.vb` or `Automobile.cs` file is there. In the code editor, the focus is still in `DefaultAutomobileIsInitializedCorrectly`, which enables you to continue to write your test with a minimum of interruption.

To generate a property stub

Assume that the product specification states that the `Automobile` class has two public properties named `Model` and `TopSpeed`. These properties must be initialized with default values of "Not specified" and -1 by the default constructor. The following unit test will verify that the default constructor sets the properties to their correct default values.

1. Add the following line of code to the `DefaultAutomobileIsInitializedCorrectly` test method.

```
Assert.IsTrue((myAuto.Model == "Not specified") && (myAuto.TopSpeed == -1));
```

```
Assert.IsTrue((myAuto.Model = "Not specified") And (myAuto.TopSpeed = -1))
```

2. Because the code references two undefined properties on `Automobile`, a wavy underline appears under `Model` and `TopSpeed`. Hover over `Model` and choose the Quick Actions light bulb, then choose **Generate property 'Automobile.Model'**.

3. Generate a property stub for the `TopSpeed` property in the same way.

In the `Automobile` class, the types of the new properties are correctly inferred from the context.

To generate a stub for a new constructor

Now we'll create a test method that will generate a constructor stub to initialize the `Model1` and `TopSpeed` properties. Later, you'll add more code to complete the test.

1. Add the following additional test method to your `AutomobileTest` class.

```
[TestMethod]
public void AutomobileWithModelNameCanStart()
{
    string model = "Contoso";
    int topSpeed = 199;
    Automobile myAuto = new Automobile(model, topSpeed);
}
```

```
<TestMethod()> Public Sub AutomobileWithModelNameCanStart()
    Dim model As String = "Contoso"
    Dim topSpeed As Integer = 199
    Dim myAuto As New Automobile(model, topSpeed)
End Sub
```

2. Click the Quick Actions light bulb under the red squiggle, and then click **Generate constructor in 'Automobile'**.

In the `Automobile` class file, notice that the new constructor has examined the names of the local variables that are used in the constructor call, found properties that have the same names in the `Automobile` class, and supplied code in the constructor body to store the argument values in the `Model1` and `TopSpeed` properties.

3. After you generate the new constructor, a wavy underline appears under the call to the default constructor in `DefaultAutomobileIsInitializedCorrectly`. The error message states that the `Automobile` class has no constructor that takes zero arguments. To generate an explicit default constructor that does not have parameters, click the Quick Actions light bulb, and then click **Generate constructor in 'Automobile'**.

To generate a stub for a method

Assume that the specification states that a new `Automobile` can be put into a Running state if its `Model1` and `TopSpeed` properties are set to something other than the default values.

1. Add the following lines to the `AutomobileWithModelNameCanStart` method.

```
myAuto.Start();
Assert.IsTrue(myAuto.IsRunning == true);
```

```
myAuto.Start()
Assert.IsTrue(myAuto.IsRunning = True)
```

2. Click the Quick Actions light bulb for the `myAuto.Start` method call and then click **Generate method 'Automobile.Start'**.

3. Click the Quick Actions light bulb for the `IsRunning` property and then click **Generate property 'Automobile.IsRunning'**.

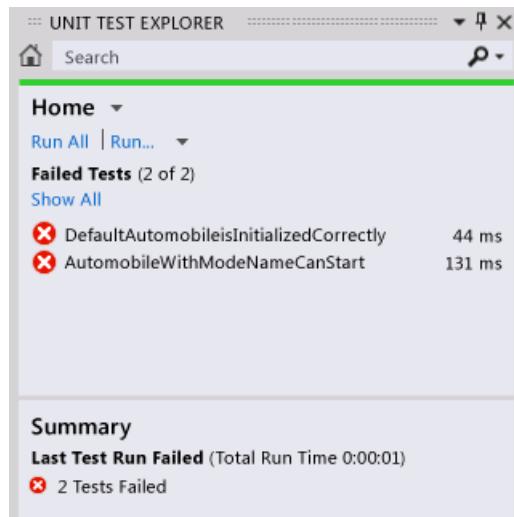
The `Automobile` class now contains a method named `start()` and a property named `IsRunning`.

To run the tests

1. On the **Test** menu, choose **Run, All Tests**.

The **Run, All Tests** command runs all the tests in any test frameworks that are written for the current solution. In this case, there are two tests, and they both fail, as expected. The `DefaultAutomobileIsInitializedCorrectly` test fails because the `Assert.IsTrue` condition returns `False`. The `AutomobileWithModelNameCanStart` test fails because the `Start` method in the `Automobile` class throws an exception.

The **Test Results** window is shown in the following illustration.



2. In the **Test Results** window, double-click on each test result row to go to the location of each test.

To implement the source code

1. Add the following code to the default constructor so that the `Model`, `TopSpeed` and `IsRunning` properties are all initialized to their correct default values of "Not specified", -1, and `False` (or `false` for C#).

```
public Automobile()
{
    this.Model = "Not specified";
    this.TopSpeed = -1;
    this.IsRunning = false;
}
```

```
Sub New()
    Model = "Not specified"
    TopSpeed = -1
    IsRunning = False
End Sub
```

2. When the `Start` method is called, it should set the `IsRunning` flag to true only if the `Model` or `TopSpeed` properties are set to something other than their default value. Remove the `NotImplementedException` from the method body and add the following code.

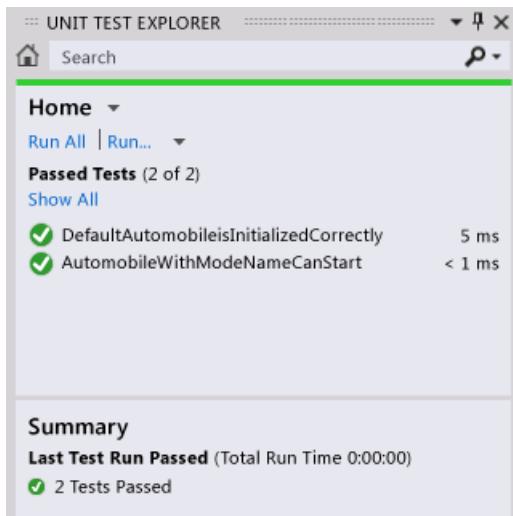
```
public void Start()
{
    if (this.Model != "Not specified" || this.TopSpeed != -1)
        this.IsRunning = true;
    else
        this.IsRunning = false;
}
```

```
Sub Start()
    If Model <> "Not specified" Or TopSpeed <> -1 Then
        IsRunning = True
    Else
        IsRunning = False
    End If
End Sub
```

To run the tests again

- On the **Test** menu, point to **Run**, and then click **All Tests**.

This time the tests pass. The **Test Results** window is shown in the following illustration.



See Also

[Generate From Usage](#)

[Writing Code](#)

[Using IntelliSense](#)

[Unit Test Your Code](#)

[Quick Actions](#)

Productivity tips for Visual Studio

3/5/2018 • 8 min to read • [Edit Online](#)

This topic contains various tips to help you write, navigate, and debug your code more quickly and efficiently.

For information about common keyboard shortcuts, see [Keyboard tips](#). Or, for a complete list of keyboard shortcuts, see [Identifying and customizing keyboard shortcuts](#) and [Default keyboard shortcuts](#).

Writing code

Write code more quickly by using the following features.

- **Use convenience commands.** Visual Studio contains various commands to help you accomplish common editing tasks faster. For example, in **Visual Studio 2017 version 15.6** and later, you can choose a command to easily duplicate a line of code without having to copy it, reposition the cursor, and then paste it. Choose **Edit > Duplicate** or press **Ctrl+E,V**. You can also quickly expand or contract a selection of text by choosing **Edit > Advanced > Expand Selection** or **Edit > Advanced > Contract Selection**, or by pressing **Shift+Alt+=** or **Shift+Alt+-** (available in **Visual Studio 2017 version 15.5** and later).
- **Use IntelliSense.** As you enter code in the editor, IntelliSense information, such as List Members, Parameter Info, Quick Info, Signature Help, and Complete Word, appears. These features support fuzzy matching of text; for example, the results lists for List Members includes not only entries that start with the characters that you have entered but also entries that contain the character combination anywhere in their names. For more information, see [Using IntelliSense](#).
- **Change auto-insertion of IntelliSense options as you enter code.** By switching IntelliSense to suggestion mode, you can specify that IntelliSense options are inserted only if you explicitly choose them.

To enable suggestion mode, choose the **Ctrl+Alt+Spacebar** keys, or, on the menu bar, choose **Edit > IntelliSense > Toggle Completion Mode**.

- **Use code snippets.** You can use built-in snippets, or create your own snippets.

To insert a snippet, on the menu bar, choose **Edit > IntelliSense > Insert Snippet** or **Surround With**, or open the shortcut menu in a file and choose **Snippet > Insert Snippet** or **Surround With**. For more information, see [Code Snippets](#).

- **Fix code errors inline.** Quick Actions let you easily refactor, generate, or otherwise modify code with a single action. These actions can be applied using the Light Bulb icon , or by pressing **Alt+Enter** or **Ctrl+.** when your cursor is on the appropriate line of code. See [Quick Actions](#) for more information.
- **Show and edit the definition of a code element.** You can quickly show and edit the module in which a code element, such as a member, a variable, or a local, is defined.

To open a definition in a pop-up window, highlight the element and then choose the **Alt+F12** keys, or open the shortcut menu for the element and then choose **Peek Definition**. To open a definition in a separate code window, open the shortcut menu for the element, and then choose **Go to Definition**.

- **Use sample applications.** You can speed up application development by downloading and installing sample applications from [Microsoft Developer Network](#). You can also learn a particular technology or programming concept by downloading and exploring a Sample Pack for that area.

Navigating within your code

You can use various techniques to find and move to specific locations in your code more quickly.

- **Bookmark lines of code.** You can use bookmarks to navigate quickly to specific lines of code in a file.

To set a bookmark, on the menu bar, choose **Edit > Bookmarks > Toggle Bookmark**. You can view all of the bookmarks for a solution in the **Bookmarks** window. For more information, see [Setting Bookmarks in Code](#).

- **Search for symbol definitions in a file.** You can search within a solution to locate symbol definitions and file names, but search results don't include namespaces or local variables.

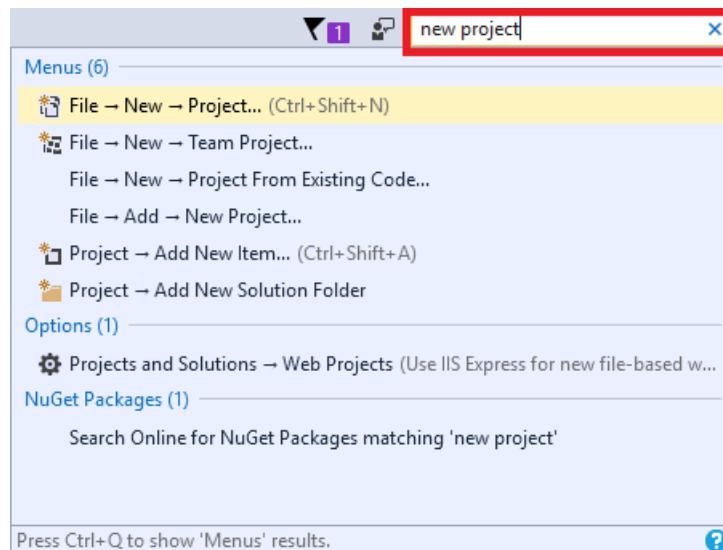
To access this feature, on the menu bar, choose **Edit > Navigate To**.

- **Browse the overall structure of your code.** In **Solution Explorer**, you can search and browse classes and their types and members in your projects. You can also search for symbols, view a method's Call Hierarchy, find symbol references, and perform other tasks. If you choose a code element in **Solution Explorer**, the associated file opens in a **Preview** tab, and the cursor moves to the element in the file. For more information, see [Viewing the Structure of Code](#).

Finding items faster

You can search across the IDE for commands, files, and options, in addition to filtering the contents of tool windows to show only relevant information for your current task.

- **Filter the contents of tool windows.** You can search within the contents of many tool windows, such as the **Toolbox**, the **Properties** window, and **Solution Explorer**, but display only items whose names contain the characters that you specify.
- **Display only the errors you want to address.** If you choose the **Filter** button on the **Error List** toolbar, you can reduce the number of errors that appear in the **Error List** window. You can display only the errors in the files that are open in the editor, only the errors in the current file, or only the errors in the current project. You can also search within the Error List window to find specific errors.
- **Find dialog boxes, menu commands, and options.** In the **Quick Launch** box, enter keywords or phrases for the items that you're trying to find. For example, the following options appear if you enter `new project`:

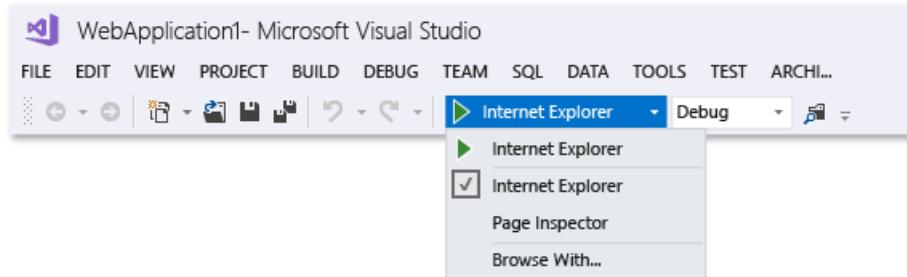


Quick Launch displays links to the **New Project** dialog box, the **Add New Item** dialog box, and the **Projects and Solutions** page in the **Options** dialog box, among others. Quick Launch results can also include project files and tool windows.

Debugging code

Debugging can consume a lot of time, but the following tips can help you speed up the process.

- **Test the same page, application, or site in different browsers.** As you debug your code, you can easily switch among the installed web browsers, including [Page Inspector \(Visual Studio\)](#), without having to open the **Browse With** dialog box. You can use the **Debug Target** list, which is on the **Standard** toolbar next to the **Start Debugging** button, to quickly verify which browser you're using as you debug or view pages.



- **Set temporary breakpoints.** You can create a temporary breakpoint in the current line of code and start the debugger simultaneously. When you hit that line of code, the debugger enters break mode. For more information, see [Navigating through Code with the Debugger](#).

To use this feature, choose the **Ctrl+F10** keys, or open the shortcut menu for the line of code on which you want to break, and then choose **Run To Cursor**.

- **Move the execution point during debugging.** You can move the current execution point to a different section of code and then restart debugging from that point. This technique is useful if you want to debug a section of code without having to recreate all of the steps that are required to reach that section. For more information, see [Navigating through Code with the Debugger](#).

To move the execution point, drag the yellow arrowhead to a location where you want to set the next statement in the same source file, and then choose the **F5** key to continue debugging.

- **Capture value information for variables.** You can add a DataTip to a variable in your code and pin it so that you can access the last known value for the variable after debugging has finished. For more information, see [View data values in Data Tips](#).

To add a DataTip, the debugger must be in break mode. Place the cursor on the variable, and then choose the pin button on the DataTip that appears. When debugging is stopped, a blue pin icon appears in the source file next to the line of code that contains the variable. If you point to the blue pin, the value of the variable from the most recent debugging session appears.

- **Clear the Immediate window.** You can erase the contents of the [Immediate Window](#) at design time by entering `>cls` or `>Edit.ClearAll`

For more information about additional commands, see [Visual Studio Command Aliases](#).

Accessing Visual Studio tools

You can quickly access the Developer Command Prompt, or another Visual Studio tool, if you pin it to the Start menu or the taskbar.

1. In Windows Explorer, browse to

```
%ProgramData%\Microsoft\Windows\Start Menu\Programs\Visual Studio 2017\Visual Studio Tools
```

2. Right-click or open the context menu for **Developer Command Prompt**, and then choose **Pin to Start** or **Pin to taskbar**.

Managing files, toolbars, and windows

At any one time, you may be working in multiple code files and moving among several tool windows as you develop an application. You can keep organized by using the following tips.

- **Keep files that you frequently use visible in the editor.** You can pin files to the left side of the tab well so that they remain visible regardless of how many files are open in the editor.

To pin a file, choose the file's tab, and then choose the **Toggle Pin Status** button.

- **Move documents and windows to other monitors.** If you use more than one monitor when you develop applications, you can work on portions of your application more easily by moving files that are open in the editor to another monitor. You can also move tool windows, such as debugger windows, to another monitor and tab dock document and tool windows together to create "rafts." For more information, see [Customize window layouts in Visual Studio](#).

You can also manage files more easily by creating another instance of **Solution Explorer** and moving it to another monitor. To create another instance of **Solution Explorer**, open a shortcut menu in **Solution Explorer**, and then choose **New Solution Explorer View**.

- **Customize the fonts that appear in Visual Studio.** You can change the font face, size, and color that's used for text in the IDE. For example, you can customize the color of specific code elements in the editor and the font face in tool windows or throughout the IDE. For more information, see [How to: Change Fonts and Colors](#) and [How to: Change Fonts and Colors in the Editor](#).

See also

- [Default Keyboard Shortcuts for Frequently Used Commands](#)
- [How to: Customize Menus and Toolbars](#)
- [Walkthrough: Create a Simple Application](#)
- [Accessibility Tips and Tricks](#)

Keyboard tips for Visual Studio

3/5/2018 • 2 min to read • [Edit Online](#)

You can navigate in Visual Studio more easily by using the keyboard shortcuts in this article.

The shortcuts listed here are only a subset of the available keyboard shortcuts. For a more complete list, see [Default Keyboard Shortcuts in Visual Studio](#).

For information about how to optimize Visual Studio for accessibility, see [Accessibility Tips and Tricks](#).

Window management

| | |
|-----------------------------|-----------------------------|
| Drag Off Floating Tab Wells | Ctrl+click for multi-select |
| Maximize Floating Window | Double-click on title bar |
| Re-dock Floating Window | Ctrl+double-click title bar |
| Close Active Document | Ctrl+F4 |
| Show Open File List | Ctrl+Alt+Down |
| Show All Floating Windows | Ctrl+Shift+M |

Window shortcuts

| | |
|----------------------------|----------------------|
| Move/Dock Floating Windows | Win+Left / Win+Right |
| Maximize/Minimize Windows | Win+Up / Win+Down |
| Show Jump List | Win+Alt+n |
| Start New Instance | Win+Shift+n |
| Switch Between Windows | Win+n |

Visual Studio search

| | |
|--|--------------------------------------|
| Solution Explorer Search | Ctrl+; |
| Place Focus in Search box in any tool window | Alt+` when the tool window has focus |
| Quick Launch | Ctrl+Q |

| | |
|----------------------------|---|
| Quick Launch Scope Results | - @opt Options - @cmd Commands - @mru Most recently used - @doc Open documents |
| Search in Tools Options | Ctrl+E |

Editor find

| | |
|-----------------------------|--------------|
| Quick Find | Ctrl+F |
| Quick Find Next Result | Enter |
| Quick Find Previous Result | Shift+Enter |
| Quick Find Expand Drop Down | Alt+Down |
| Dismiss Find | Esc |
| Quick Replace | Ctrl+H |
| Quick Replace Replace Next | Alt+R |
| Quick Replace Replace All | Alt+A |
| Find in Files | Ctrl+Shift+F |
| Replace in Files | Ctrl+Shift+H |

Code Editor

| COMMAND | SHORTCUT | VERSION AVAILABILITY |
|---------------------------------|---|----------------------|
| IntelliSense Suggestion Mode | Ctrl+Alt+Space (Toggle) | |
| Force Show IntelliSense | Ctrl+J | |
| Quick Actions | Ctrl+. | |
| Snippet Picker | Ctrl+K,X or ?,Tab (VB) | |
| Surround With | Ctrl+K,S | |
| Show Quick Info | Ctrl+K,I | |
| Navigate To | Ctrl+, | |
| Navigate Highlighted References | Ctrl+Shift+Up (Previous), Ctrl+Shift+Down (Next) | |

| COMMAND | SHORTCUT | VERSION AVAILABILITY |
|---|--|---------------------------------|
| Editor Zoom | Ctrl+Shift+> (In), Ctrl+Shift+< (Out) | |
| Block Selection | Hold Alt and drag mouse, Shift+Alt+Arrow Keys | |
| Move Line Up/Down | Alt+Up / Alt+Down | |
| Duplicate line | Ctrl+E,V | Visual Studio 2017 version 15.6 |
| Expand selection | Shift+Alt+= | Visual Studio 2017 version 15.5 |
| Contract selection | Shift+Alt+- | Visual Studio 2017 version 15.5 |
| Go To Definition | F12 | |
| Peek Definition | Alt+F12 | |
| Go To Definition Stack | Ctrl+Shift+8 (Back), Ctrl+Shift+7 (Forward) | |
| Close the Peek Definition window | Esc | |
| Promote the Peek Definition window to a regular document tab | Ctrl+Alt+Home | |
| Navigate between multiple Peek Definition windows | Ctrl+Alt+- and Ctrl+Alt+= | |
| Navigate between multiple Peek results | F8 and Shift+F8 | |
| Toggle between the code editor window and the Peek Definition window | Shift+Esc | |

Toolbars

| | |
|--------------------------------|-------------------------------|
| Add Buttons | Click toolbar overflow button |
| Find Combo in Standard toolbar | Ctrl+D |
| Find Textbox Command Mode | Type ">" |
| Create new alias | >alias NewAlias Command |

Debugging

| | |
|-----------------|----|
| Start Debugging | F5 |
|-----------------|----|

| | |
|-------------------------------|----------------|
| Stop Debugging | Shift+F5 |
| Restart Debugging | Ctrl+Shift+F5 |
| Step Over | F10 |
| Step Into | F11 |
| Step Out | Shift+F11 |
| Run To Cursor | Ctrl+F10 |
| Set Next Statement | Ctrl+Shift+F10 |
| Set and Toggle Breakpoint | F9 |
| Disable Breakpoint | Ctrl+F9 |
| Immediate Window | Ctrl+Alt+I |
| Immediate Window Command Mode | Type ">" |
| Immediate Window Clear Buffer | >cls |
| Immediate Window Print Value | ?varname |

See also

- [Keyboard shortcuts \(VSTS and TFS\)](#)
- [Visual Studio Blog](#)
- [Visual Studio Tips and Tricks Blog](#)
- [Visual Studio Toolbox on Channel 9](#)
- [Visual Studio UserVoice](#)
- [Visual Studio Connect Bugs](#)

Visual Studio 2017 Productivity Guide for .NET Developers

3/27/2018 • 7 min to read • [Edit Online](#)

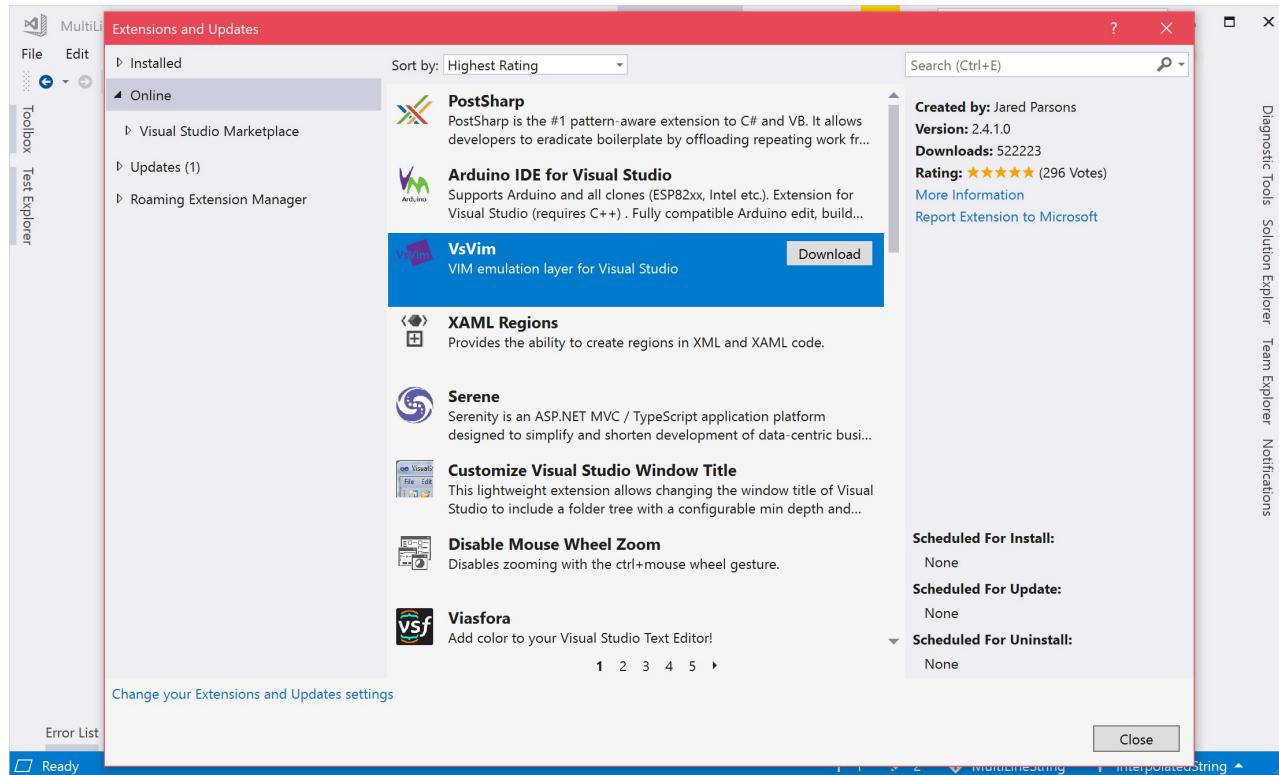
Visual Studio 2017 makes developers more productive than ever! We improved performance and reliability for solution startup and load, test discovery, and typing latency. We've also added and enhanced features to help you write better code faster. Some of these features include: navigation to decompiled assemblies, variable name suggestions as you type, a hierarchy-view in the Test Explorer, Go To All (**Ctrl+T**) to navigate to file/type/member/symbol declarations, an intelligent Exception Helper, code style configuration and enforcement, and many refactorings and code fixes.

Follow this guide to optimize your productivity.

I'm used to my keyboard shortcuts from a different extension/editor/IDE.

If you are coming from another IDE or coding environment, you may find installing one of these extensions helpful:

- [Emacs Emulation](#)
- [HotKeys for Visual Studio \(ReSharper/IntelliJ\)](#)
- [VSVim](#)



The following are popular Visual Studio shortcuts.

NOTE

Some extensions unbind default Visual Studio keybindings so you must restore them to use the following commands. Restore your keybindings to Visual Studio's defaults by going to: **Tools > Import and Export Settings... > Reset all settings** or **Tools > Options > Keyboard > Reset**.

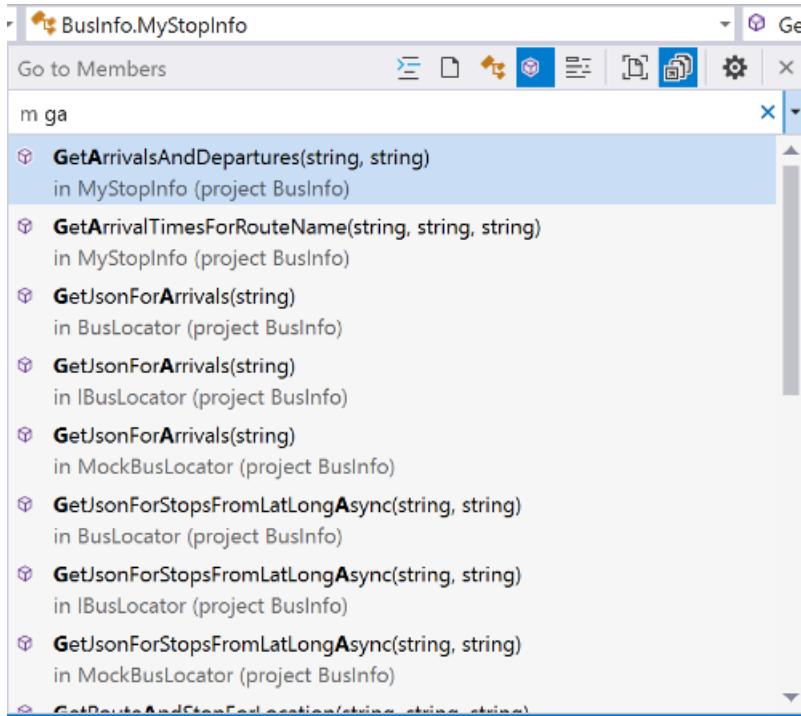
| SHORTCUT (ALL PROFILES) | COMMAND | DESCRIPTION |
|---|--------------------------------|--|
| Ctrl+T | Go To All | Navigate to any file/type/member/symbol declaration |
| F12 (also Ctrl+Click) | Go To Definition | Navigate to where a symbol is defined |
| Ctrl+F12 | Go To Implementation | Navigate from a base type or member to its various implementations |
| Shift+F12 | Find All References | See all symbol or literal references |
| Ctrl+. (also Alt+Enter in C# Profile) | Quick Actions and Refactorings | See what code fixes, code generation actions, refactorings, or other quick actions are available at your cursor position or code selection |
| Ctrl+D | Duplicate line | Duplicates the line of code that the cursor is in (available in Visual Studio 2017 version 15.6 and later) |
| Shift+Alt++/- | Expand/Contract selection | Expands or contracts the current selection in the editor (available in Visual Studio 2017 version 15.5 and later) |
| Ctrl+Q | Quick Launch | Search all Visual Studio settings |
| F5 | Start Debugging | Start debugging your application |
| Ctrl+F5 | Run without Debug | Run your application locally without debugging |
| Ctrl+K,D (Default Profile) or Ctrl+E,D (C# Profile) | Format Document | Cleans up formatting violations in your file based on your newline, spacing, and indentation settings |
| Ctrl+\E (Default Profile) or Ctrl+W,E (C# Profile) | View Error List | See all errors in your document, project, or solution |

I need a way to quickly navigate to files or types.

Visual Studio 2017 has a feature called *Go To All* (**Ctrl+T**). Go To All enables you to quickly jump to any file, type, member, or symbol declaration.

- Change the location of this search bar or turn off the 'live navigation preview' with the **gear** icon
- Filter results using our query syntax (for example, "t mytype"). You can also scope your search to just the current document.

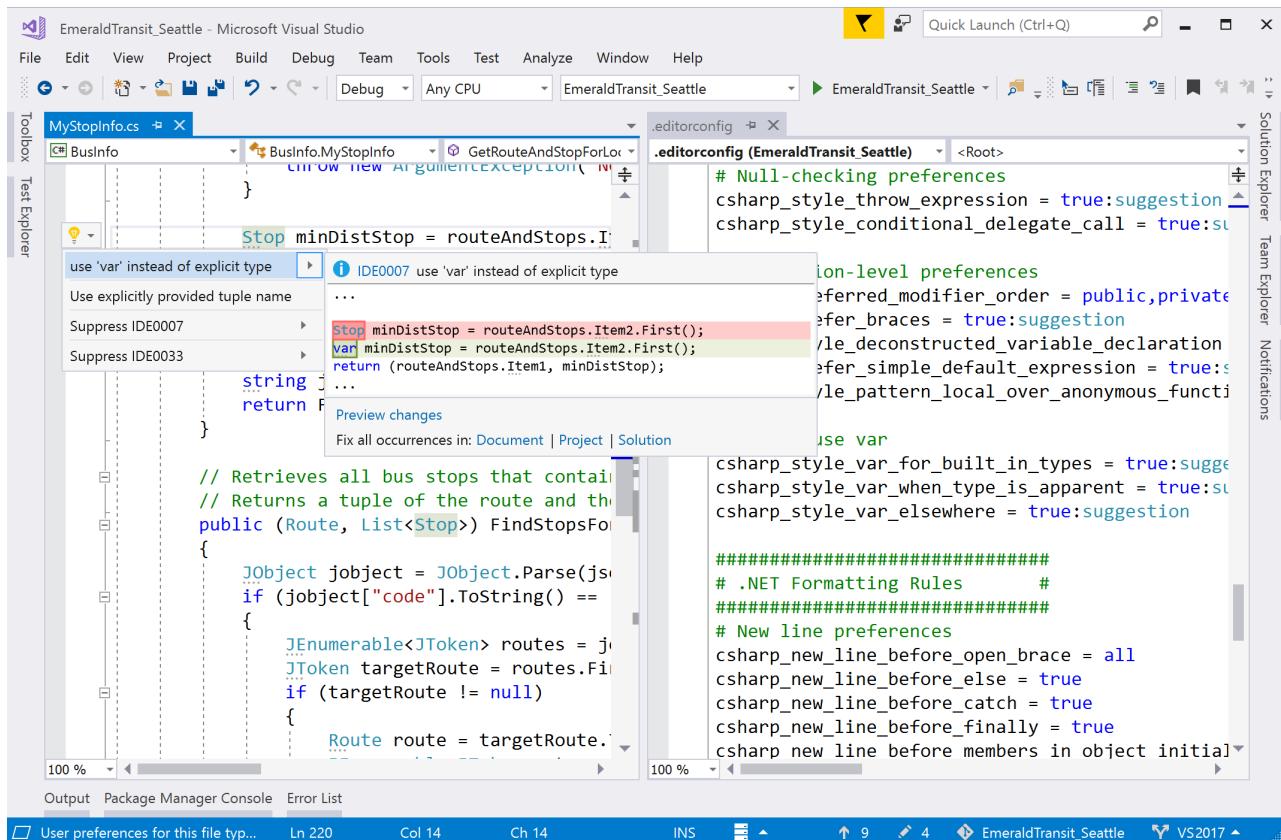
- camelCase matching is supported!



My team enforces code style rules on our codebase.

You can use an `.editorconfig` file to codify coding conventions and have them travel with your source.

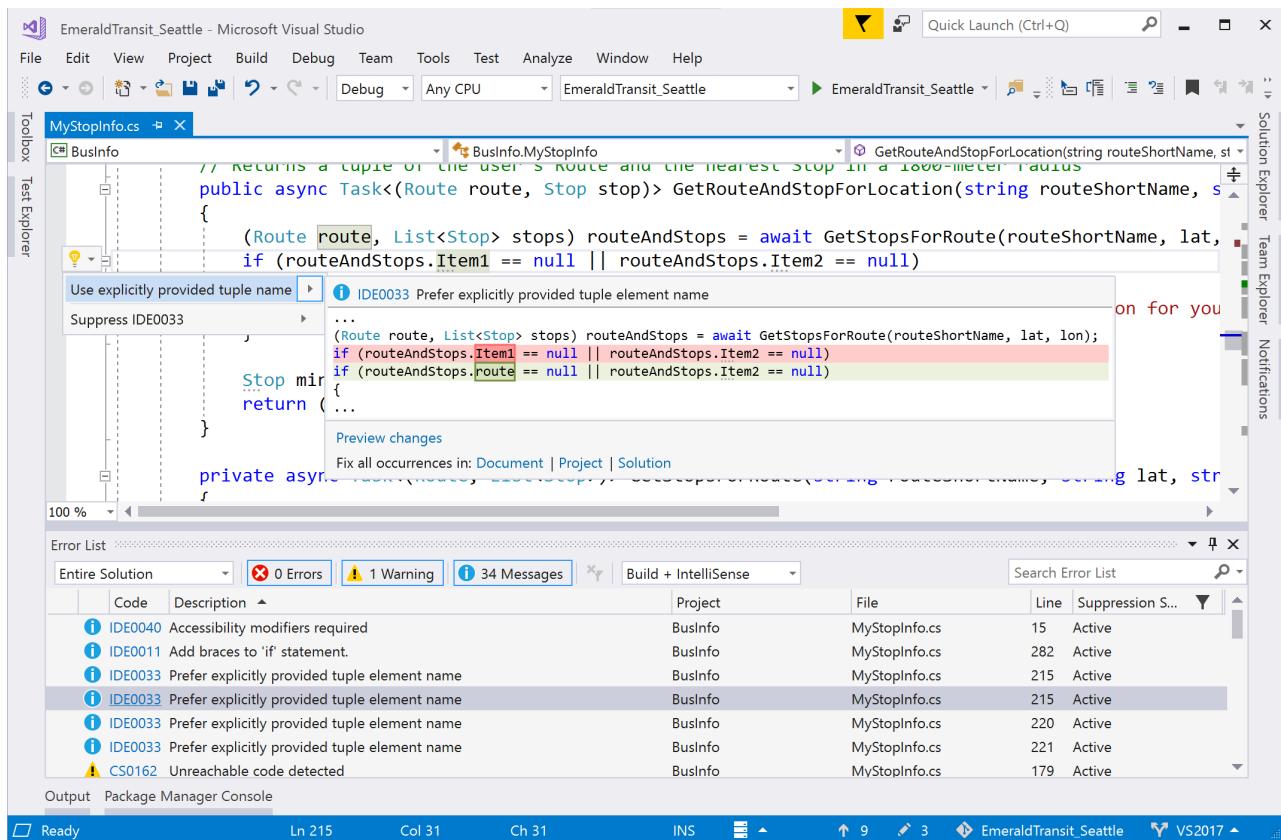
- We recommend installing the [EditorConfig Language Services extension](#) for adding and editing an `.editorconfig` file in Visual Studio.
- Check out the [documentation](#) for all .NET coding convention options.
- See [this gist](#) for an example `.editorconfig`.



I need more refactorings and code fixes.

Visual Studio 2017 comes with a lot of refactorings, code generation actions, and code fixes. Red squiggles represent errors, green squiggles represent warnings, and three gray dots represent code suggestions. You can access code fixes by clicking the lightbulb/screwdriver icon or by pressing **Ctrl+.** or **Alt+Enter**. Each fix comes with a preview window that shows a live code diff of how the fix works.

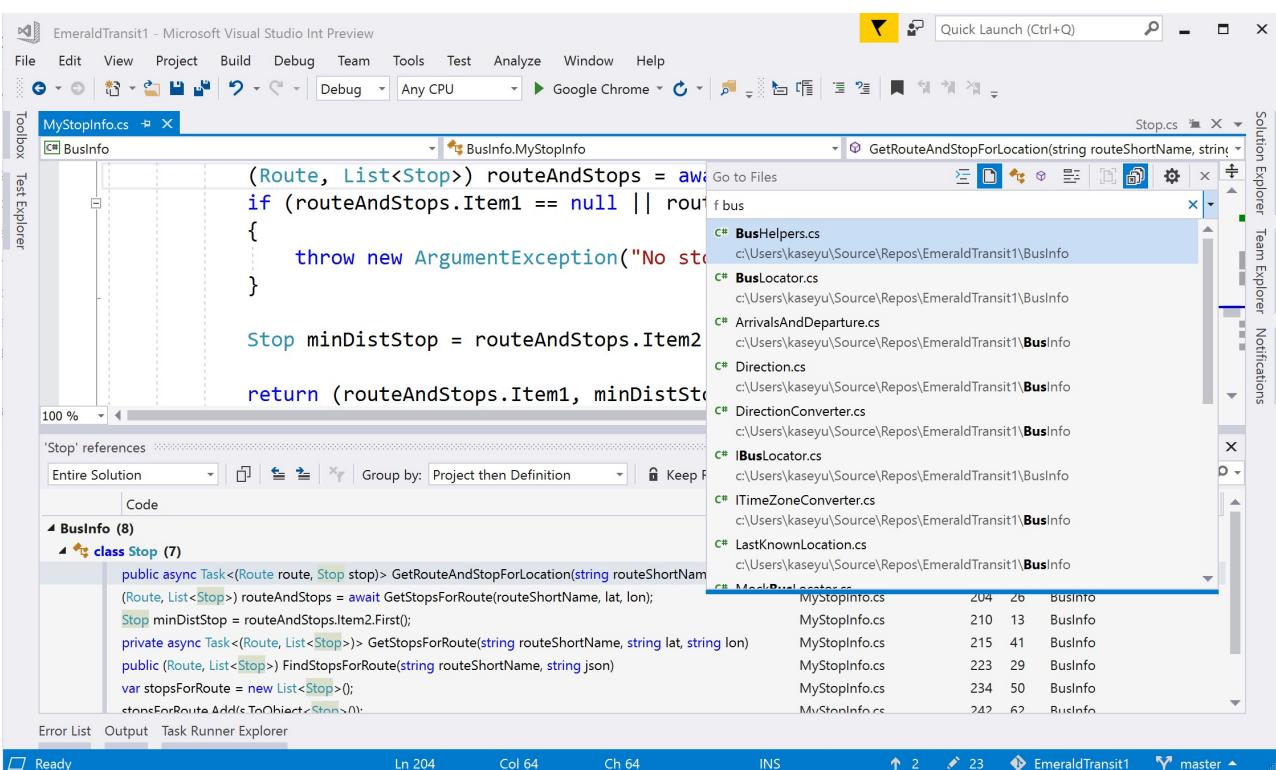
- Popular quick fixes and refactorings include:
 - *Rename*
 - *Extract Method*
 - *Change Method Signature*
 - *Generate Constructor*
 - *Generate Method*
 - *Move Type to File*
 - *Add Null-Check*
 - *Add Parameter*
 - *Remove Unnecessary Usings*
 - See more in our [documentation](#)
- Write your own refactoring or code fix with [Roslyn analyzers](#).
- Several community members have written *free* extensions which add additional code inspections:
 - [Roslynator](#)
 - [SonarLint for Visual Studio](#)
 - [StyleCopAnalyzers](#)



I need Find Usages, Go To Implementation, Navigate To Decompiled Assemblies

Visual Studio 2017 has many features to help you search and navigate your codebase. Read more about [code navigation features](#)

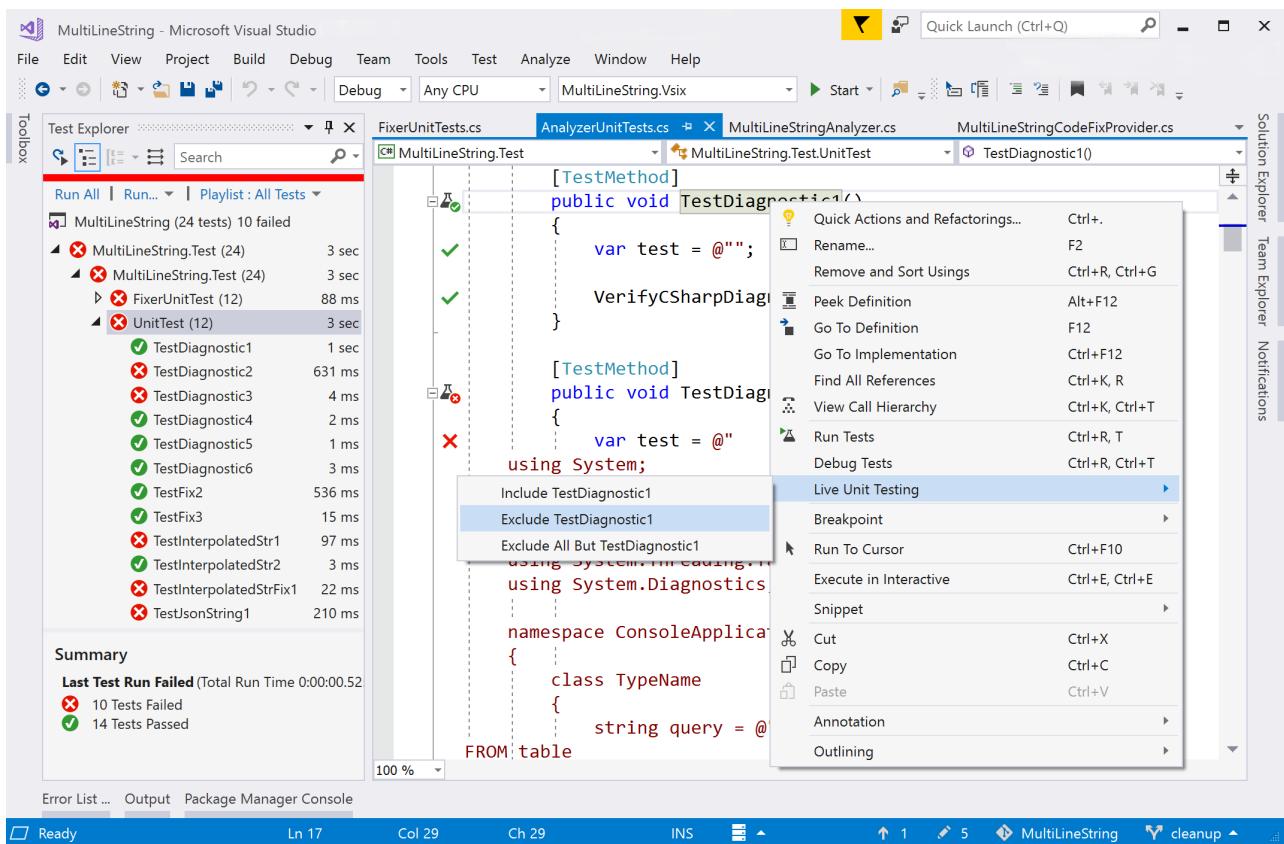
| FEATURE | SHORTCUT | DETAILS/IMPROVEMENTS |
|-------------------------------------|-----------------------------------|--|
| Find All References | Shift+F12 | Results are colorized and can be grouped by project, definition, etc. You can also 'lock' results. |
| Go To Implementation | Ctrl+F12 | You can use Go To Definition on the <code>override</code> keyword to navigate to the overridden member |
| Go To Definition | F12 or Ctrl+Click | You can hold Ctrl while clicking to navigate to definition |
| Peek Definition | Alt+F12 | Inline view of a definition |
| Structure Visualizer | Gray, dotted-lines between braces | Hover to see your code structure |
| Navigation to decompiled assemblies | F12 or Ctrl+Click | Navigate to external source (decompiled with ILSpy) by enabling the feature: Tools > Options > Text Editor > C# > Advanced > Enable navigation to decompiled sources. |



I want to run and see my unit tests.

We made a lot of improvements to the testing experience in Visual Studio 2017. Use either of our unit testing experiences with the MSTest v1, MSTest v2, NUnit, or XUnit test frameworks.

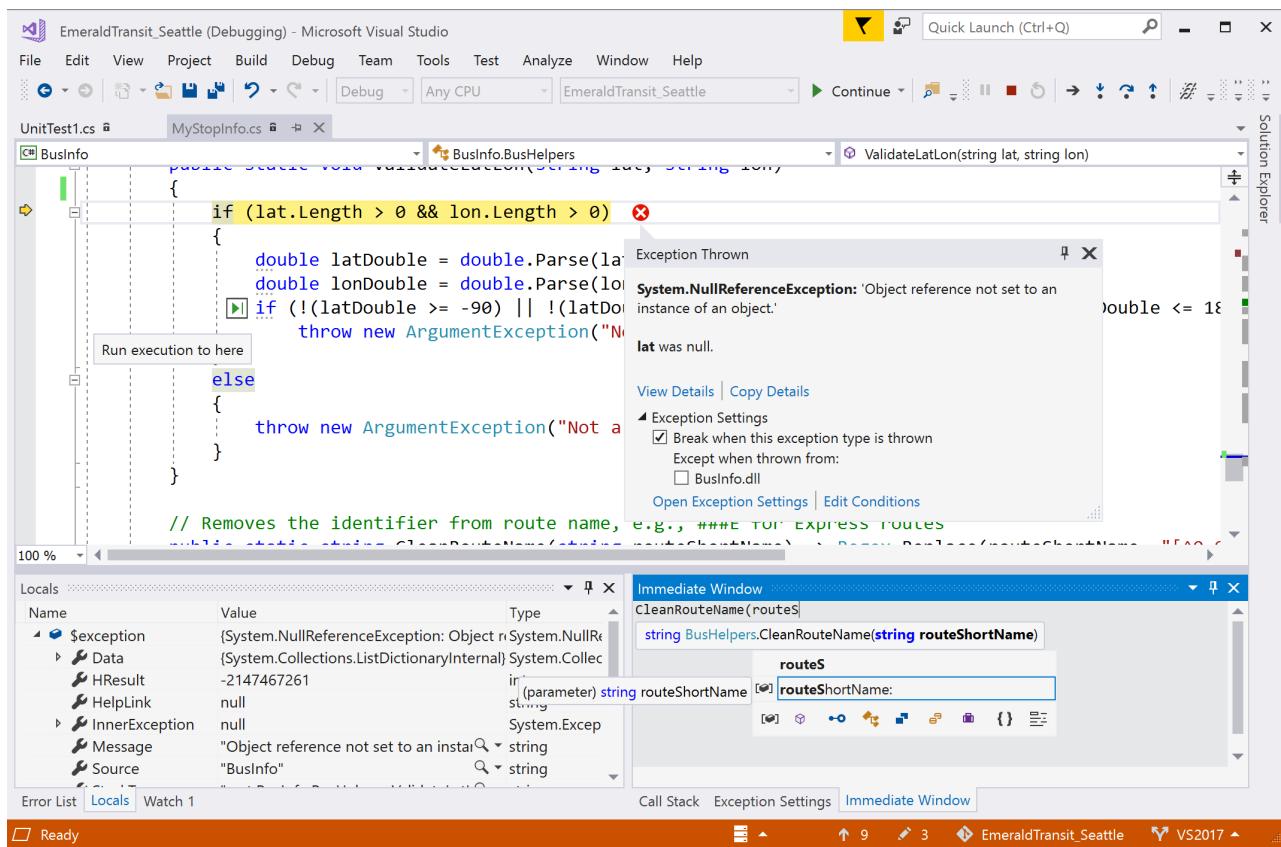
- *Test Explorer* test discovery is fast in version 15.6 (for best results, upgrade to the latest version of your test adapter).
- Organize your tests in Test Explorer with our new *hierarchical sorting* in version 15.6.
- *Live Unit Testing* continuously runs tests impacted by your code changes and updates inline editor icons to let you know the status of your tests. Include or exclude specific tests or test projects from your *Live Test Set*.



I want to debug my code.

We've added a ton of new debugging capabilities in Visual Studio 2017.

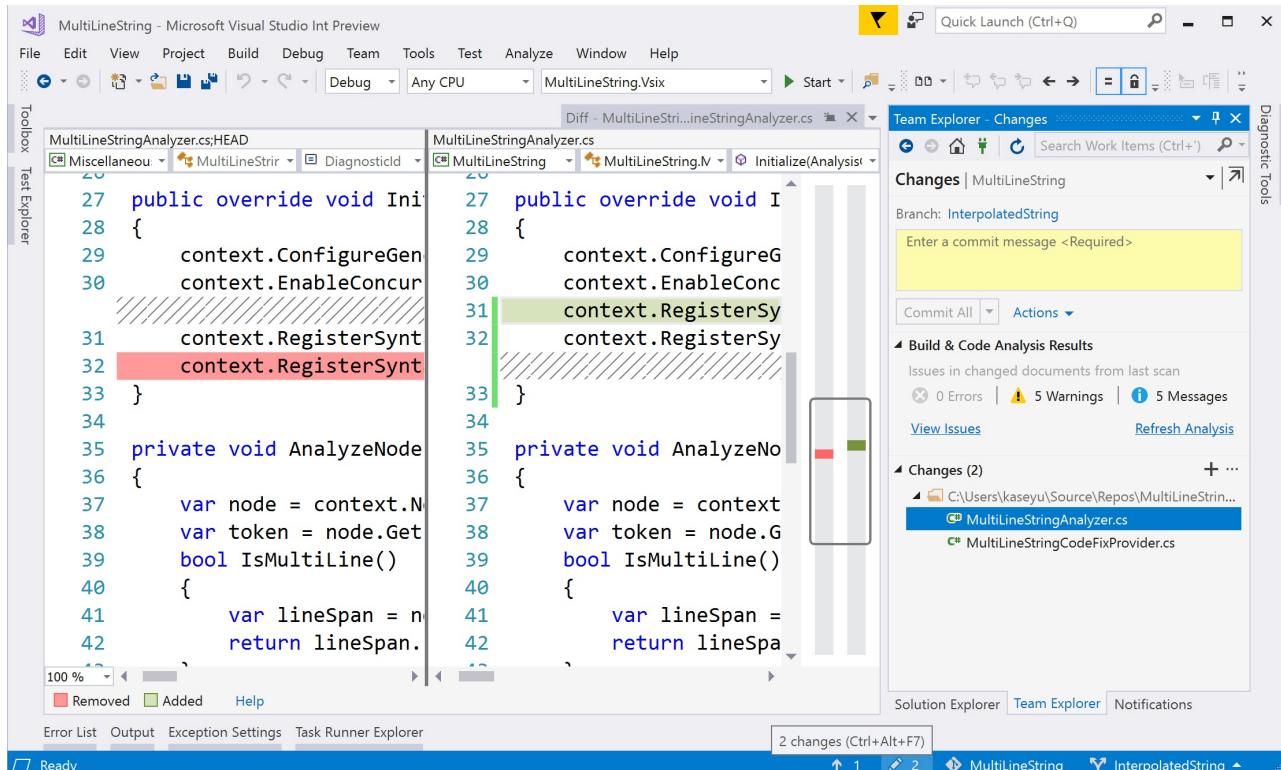
- *Run to click* allows you to hover next to a line of code, hit the green 'play' icon that appears, and run your program until it reaches that line.
- The new *Exception Helper* puts the most important information, like which variable is 'null' in a `NullReferenceException`, at the top-level in the dialog.
- *Step Back* debugging enables you to go back to previous breakpoints or steps and view the state of the application as it was in the past.
- *Snapshot Debugging* lets you investigate the state of a live web application at the moment an exception was thrown (must be on Azure).



I want to use version control with my projects.

You can use git or TFVC to store and update your code in Visual Studio.

- Organize your local changes with *Team Explorer* and use the status bar to track pending commits and changes.
- Set up continuous integration and delivery for your projects inside of Visual Studio with our [Continuous Delivery Tools for Visual Studio](#) extension and adopt the agile developer workflow.



What other features do I need to know about?

Here is a list of editor and productivity features to make writing code more efficient. Some features may need to be enabled because they are off-by-default (they may index things on your machine, are controversial, or are currently experimental).

| FEATURE | DETAILS | HOW TO ENABLE |
|---|--|---|
| Locate File in Solution Explorer | Highlights the active file in the Solution Explorer | Tools > Options > Projects and Solutions > Track Active Item in Solution Explorer |
| Add usings for types in reference assemblies and NuGet packages | Shows a lightbulb with a code fix to install a NuGet package for an unreferenced type | Tools > Options > Text Editor > C# > Advanced > Suggest usings for types in reference assemblies and Suggest usings for types in NuGet packages |
| Enable full solution analysis | See all errors in your solution in the Error List | Tools > Options > Text Editor > C# > Advanced > Enable full solution analysis |
| Enable navigation to decompiled sources | Allow Go To Definition on types/members from external sources and use the ILSpy decompiler to show method bodies | Tools > Options > Text Editor > C# > Advanced > Enable navigation to decompiled sources |
| Completion/Suggestion Mode | Changes the completion behavior in IntelliSense--developers with IntelliJ backgrounds tend to change the setting here from the default | Menu > Edit > IntelliSense > Toggle Completion Mode |
| CodeLens | Displays code reference information and change history in the editor | Tools > Options > Text Editor > All Languages>CodeLens |
| Code snippets | Help stub out common boilerplate | Type a snippet name and press 'Tab' twice. |

```
class House
{
    public Color Color { get; set; }
    public int MyProperty { get; set; }
    public House(Color color, List<Person> )
}
```

Refactoring Examples - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test Analyze Window Help

NuGet: RefactoringExamples Program.cs

Solution Explorer Team Explorer Notifications

Toolbox Test Explorer

2 references

1 reference

0 references

1 reference

150 % Output Package Manager Console Find All References Error List

Ready Ln 102 Col 48 Ch 48 INS Add to Source Control

Missing a feature that makes you productive or experiencing poor performance?

There are several ways to leave us feedback:

- .NET feature requests can be filed on our [GitHub repo](#).
- Visual Studio feature requests, bugs, and performance issues can be filed by using the **Send Feedback** icon at the top-right of your Visual Studio window.

Identifying and customizing keyboard shortcuts in Visual Studio

12/22/2017 • 4 min to read • [Edit Online](#)

You can identify keyboard shortcuts for Visual Studio commands, customize those shortcuts, and export them for others to use. Many shortcuts always invoke the same commands, but the behavior of a shortcut can vary based on the following conditions:

- Which default environment settings you chose the first time that you ran Visual Studio (for example, General Development or Visual C#).
- Whether you've customized the shortcut's behavior.
- Which context you're in when you choose the shortcut. For example, the F2 shortcut invokes the `Edit.EditCell` command if you're using the Settings Designer and the `File.Rename` command if you're using Team Explorer.

Regardless of settings, customization, and context, you can always find and change a keyboard shortcut in the **Options** dialog box. You can also look up the default keyboard shortcuts for several dozen commands in [Default Keyboard Shortcuts for Frequently Used Commands](#), and you can find a complete list of all default shortcuts (based on the General Development Settings) in [Default Keyboard Shortcuts](#).

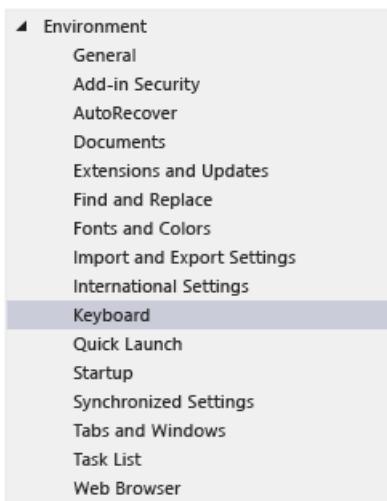
If a shortcut is assigned to a command in the Global context and no other contexts, that shortcut will always invoke that command. But a shortcut can be assigned to one command in the Global context and a different command in a specific context. If you use such a shortcut when you're in the specific context, the shortcut invokes the command for the specific context, not the Global context.

NOTE

Your settings and edition of Visual Studio might change the names and locations of menu commands and the options that appear in dialog boxes. This topic is based on the **General Development Settings**.

Identifying a keyboard shortcut

1. On the menu bar, choose **Tools**, **Options**.
2. Expand **Environment**, and then choose **Keyboard**.



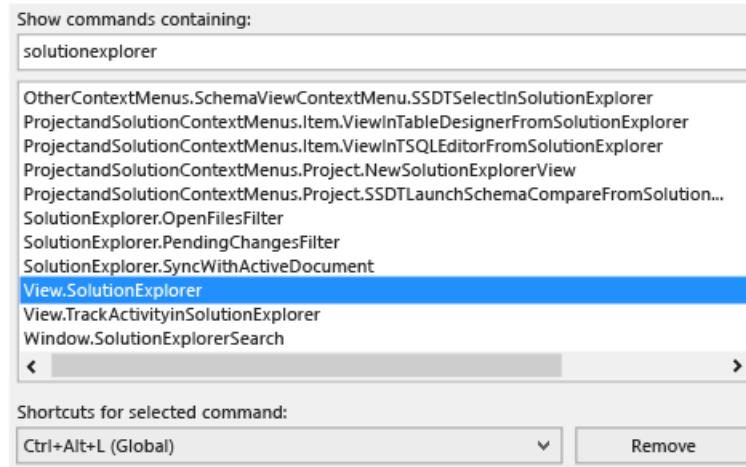
3. In the **Show commands containing** box, enter all or part of the name of the command without spaces.

For example, you can find commands for `solutionexplorer`.

4. In the list, choose the correct command.

For example, you can choose **View.SolutionExplorer**.

5. If the command has a keyboard shortcut, it appears in the **Shortcut(s) for selected command** list.



Customizing a keyboard shortcut

1. On the menu bar, choose **Tools, Options**.

2. Expand the **Environment** folder, and then choose **Keyboard**.

3. Optional: Filter the list of commands by entering all or part of the name of the command, without spaces, in the **Show commands containing** box.

4. In the list, choose the command to which you want to assign a keyboard shortcut.

In the **Use new shortcut in** list, choose the feature area in which you want to use the shortcut.

For example, you can choose ****Global**** if you want the shortcut to work in all contexts. You can use any shortcut that isn't mapped (as Global) in another editor. Otherwise, the editor overrides the shortcut.

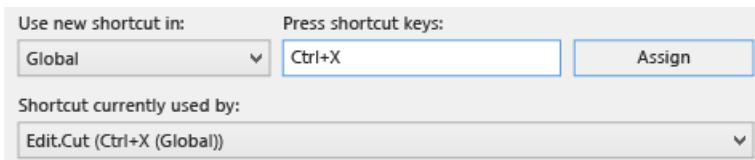
> [!NOTE]
You can't assign the following keys as part of a keyboard shortcut in ****Global****: Print Scrn/Sys Rq, Scroll Lock, Pause/Break, Tab, Caps Lock, Insert, Home, End, Page Up, Page Down, the Windows logo key, the Application key, any of the Arrow keys, or Enter; Num Lock, Delete, or Clear on the numeric keypad; the Ctrl+Alt+Delete key combination.

1. In the **Press shortcut key(s)** box, enter the shortcut that you want to use.

NOTE

You can create a shortcut that combines a letter with the Alt key, the Ctrl key, or both. You can also create a shortcut that combines the Shift key and a letter with the Alt key, the Ctrl key, or both.

If a shortcut is already assigned to another command, it appears in the **Shortcut currently used by** box. In that case, choose the Backspace key to delete that shortcut before you try a different one.



2. Choose the **Assign** button.

NOTE

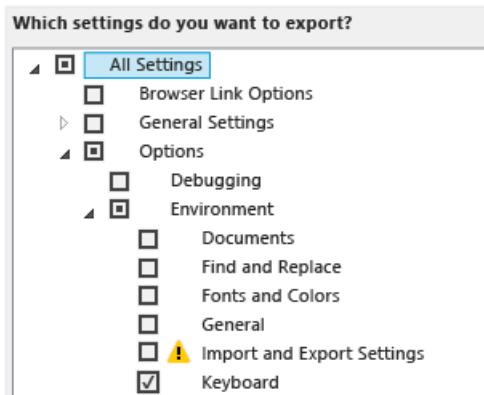
If you specify a different shortcut for a command, choose the **Assign** button, and then choose the **Cancel** button, the dialog box closes, but the change isn't reverted.

Sharing custom keyboard shortcuts

You can share your custom keyboard shortcuts by exporting them to a file and then giving the file to others so that they can import the data.

To export only keyboard shortcuts

1. On the menu bar, choose **Tools, Import and Export Settings**.
2. Choose **Export selected environment settings**, and then choose the **Next** button.
3. Under **What settings do you want to export?**, clear the **All Settings** check box, expand **Options**, and then expand **Environment**.
4. Select the **Keyboard** check box, and then choose the **Next** button.



5. In the **What do you want to name your settings file?** and **Store my settings file in this directory** boxes, either leave the default values or specify different values, and then choose the **Finish** button.

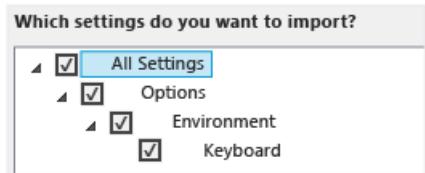
By default, your shortcuts are saved in a file in the %USERPROFILE%\Documents\Visual Studio 2017\Settings folder. The name of the file reflects the date when you exported the settings, and the extension is .vssettings.

To import only keyboard shortcuts

1. On the menu bar, choose **Tools, Import and Export Settings**.
2. Choose the **Import selected environment settings** option button, and then choose the **Next** button.
3. Choose the **No, just import new settings, overwriting my current settings** option button, and then choose the **Next** button.
4. Under **My Settings**, choose the file that contains the shortcuts that you want to import, or choose the **Browse** button to locate the correct file.
5. Choose the **Next** button.

6. Under **Which settings do you want to import?**, clear the **All Settings** check box, expand **Options**, and then expand **Environment**.

7. Select the **Keyboard** check box, and then choose the **Finish** button.



See also

[Accessibility Features of Visual Studio](#)

Default Keyboard Shortcuts for Frequently Used Commands in Visual Studio

3/5/2018 • 2 min to read • [Edit Online](#)

You can access frequently used commands in Visual Studio by choosing these default keyboard shortcuts. For a complete list of commands that have default shortcuts, see [Default Keyboard Shortcuts](#).

NOTE

You can look up the shortcut for any command by opening the **Options** dialog box, expanding the **Environment** node, and then choosing **Keyboard**.

Build

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|---------------------------------|-------------------------------|
| Build.BuildSolution | Ctrl+Shift+B [Global] |
| Build.Cancel | Ctrl+Break [Global] |
| Build.Compile | Ctrl+F7 [Global] |
| Build.RunCodeAnalysisonSolution | Alt+F11 [Global] |

Debug

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|----------------------------|---|
| Debug.BreakatFunction | Ctrl+B [Global] |
| Debug.BreakAll | Ctrl+Alt+Break [Global] |
| Debug.DeleteAllBreakpoints | Ctrl+Shift+F9 [Global] |
| Debug.Exceptions | Ctrl+Alt+E [Global] |
| Debug.QuickWatch | Ctrl+Alt+Q [Global] or Shift+F9 [Global] |
| Debug.Restart | Ctrl+Shift+F5 [Global] |
| Debug.RunWithCursor | Ctrl+F10 [Global] |
| Debug.SetNextStatement | Ctrl+Shift+F10 [Global] |
| Debug.Start | F5 [Global] |

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|-----------------------------|-------------------------------|
| Debug.StartWithoutDebugging | Ctrl+F5 [Global] |
| Debug.StepInto | F11 [Global] |
| Debug.StepOut | Shift+F11 [Global] |
| Debug.StepOver | F10 [Global] |
| Debug.StopDebugging | Shift+F5 [Global] |
| Debug.ToggleBreakpoint | F9 [Global] |

Edit

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|----------------------------|--|
| Edit.BreakLine | Enter [Text Editor, Report Designer, Windows Forms Designer] or Shift+Enter [Text Editor] |
| Edit.CollapseToDefinitions | Ctrl+M, Ctrl+O [Text Editor] |
| Edit.CommentSelection | Ctrl+K, Ctrl+C [Text Editor] |
| Edit.CompleteWord | Alt+Right Arrow [Text Editor, Workflow Designer] or Ctrl+Spacebar [Text Editor, Workflow Designer] or Ctrl+K, W [Workflow Designer] or Ctrl+K, Ctrl+W [Workflow Designer] |
| Edit.Copy | Ctrl+C [Global] or Ctrl+Insert [Global] |
| Edit.Cut | Ctrl+X [Global] or Shift+Delete [Global] |
| Edit.Delete | Delete [Global, Team Explorer] or Shift+Delete [Sequence Diagram, UML Activity Diagram, Layer Diagram] or Ctrl+Delete [Class Diagram] |
| Edit.Find | Ctrl+F [Global] |
| Edit.FindAllReferences | Shift+F12 [Global] |
| Edit.FindInFiles | Ctrl+Shift+F [Global] |

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|---------------------------|---|
| Edit.FindNext | F3 [Global] |
| Edit.FindNextSelected | Ctrl+F3 [Global] |
| Edit.FormatDocument | Ctrl+K, Ctrl+D [Text Editor] |
| Edit.FormatSelection | Ctrl+K, Ctrl+F [Text Editor] |
| Edit.GoTo | Ctrl+G [Global] |
| Edit.GoToDeclaration | Ctrl+F12 [Global] |
| Edit.GoToDefinition | F12 |
| Edit.GoToFindCombo | Ctrl+D [Global] |
| Edit.GoToNextLocation | F8 [Global] |
| Edit.InsertSnippet | Ctrl+K, Ctrl+X [Global] |
| Edit.InsertTab | Tab [Report Designer, Windows Forms Designer, Text Editor] |
| Edit.LineCut | Ctrl+L [Text Editor] |
| Edit.LineDownExtendColumn | Shift+Alt+Down Arrow [Text Editor] |
| Edit.LineOpenAbove | Ctrl+Enter [Text Editor] |
| Edit.ListMembers | Ctrl+J [Text Editor, Workflow Designer] or Ctrl+K, Ctrl+L [Workflow Designer] or Ctrl+K, L [Workflow Designer] |
| Edit.NavigateTo | Ctrl+, [Global] |
| Edit.OpenFile | Ctrl+Shift+G [Global] |
| Edit.OvertypeMode | Insert [Text Editor] |
| Edit.ParameterInfo | Ctrl+Shift+Spacebar [Text Editor, Workflow Designer] or Ctrl+K, Ctrl+P [Workflow Designer] or Ctrl+K, P [Workflow Designer] |
| Edit.Paste | Ctrl+V [Global] or Shift+Insert [Global] |
| Edit.PeekDefinition | Alt+F12 [Text Editor] |

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|-------------------------------|---|
| Edit.Redo | Ctrl+Y [Global] or Shift+Alt+Backspace [Global] or Ctrl+Shift+Z [Global] |
| Edit.Replace | Ctrl+H [Global] |
| Edit.SelectAll | Ctrl+A [Global] |
| Edit.SelectCurrentWord | Ctrl+W [Text Editor] |
| Edit.SelectionCancel | Esc [Text Editor, Report Designer, Settings Designer, Windows Forms Designer, Managed Resources Editor] |
| Edit.SurroundWith | Ctrl+K, Ctrl+S [Global] |
| Edit.TabLeft | Shift+Tab [Text Editor, Report Designer, Windows Forms Editor] |
| Edit.ToggleAllOutlining | Ctrl+M, Ctrl+L [Text Editor] |
| Edit.ToggleBookmark | Ctrl+K, Ctrl+K [Text Editor] |
| Edit.ToggleCompletionMode | Ctrl+Alt+Space [Text Editor] |
| Edit.ToggleOutliningExpansion | Ctrl+M, Ctrl+M [Text Editor] |
| Edit.UncommentSelection | Ctrl+K, Ctrl+U [Text Editor] |
| Edit.Undo | Ctrl+Z [Global] or Alt+Backspace [Global] |
| Edit.WordDeleteToEnd | Ctrl+Delete [Text Editor] |
| Edit.WordDeleteToStart | Ctrl+Backspace [Text Editor] |

File

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|-----------------|-------------------------------|
| File.Exit | Alt+F4 [Global] |
| File.NewFile | Ctrl+N [Global] |
| File.NewProject | Ctrl+Shift+N [Global] |
| File.NewWebSite | Shift+Alt+N [Global] |
| File.OpenFile | Ctrl+O [Global] |

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|------------------------|-------------------------------|
| File.OpenProject | Ctrl+Shift+O [Global] |
| File.OpenWebSite | Shift+Alt+O [Global] |
| File.Rename | F2 [Team Explorer] |
| File.SaveAll | Ctrl+Shift+S [Global] |
| File.SaveSelectedItems | Ctrl+S [Global] |
| File.ViewinBrowser | Ctrl+Shift+W [Global] |

Project

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|-------------------------|-------------------------------|
| Project.AddExistingItem | Shift+Alt+A [Global] |
| Project.AddNewItem | Ctrl+Shift+A [Global] |

Refactor

| COMMAND | KEYBOARD SHORTCUT [CONTEXT] |
|------------------------|-----------------------------|
| Refactor.ExtractMethod | Ctrl+R, Ctrl+M [Global] |

Tools

| COMMAND | KEYBOARD SHORTCUT [CONTEXT] |
|-----------------------|-----------------------------|
| Tools.AttachtoProcess | Ctrl+Alt+P [Global] |

View

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|-----------------------|--|
| View.ClassView | Ctrl+Shift+C [Global] |
| View.EditLabel | F2 [Global] |
| View.ErrorList | Ctrl+\, Ctrl+E [Global] or Ctrl+\, E [Global] |
| View.NavigateBackward | Ctrl+- [Global] |
| View.NavigateForward | Ctrl+Shift+- [Global] |

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|-----------------------|---|
| View.ObjectBrowser | Ctrl+Alt+J [Global] |
| View.Output | Ctrl+Alt+O [Global] |
| View.PropertiesWindow | F4 |
| View.Refresh | F5 [Team Explorer, Team Foundation Build Detail Editor] |
| View.ServerExplorer | Ctrl+Alt+S [Global] |
| View.ShowSmartTag | Ctrl+. [Global] or Shift+Alt+F10 [Global, HTML Editor Design View] |
| View.SolutionExplorer | Ctrl+Alt+L [Global] |
| View.TfsTeamExplorer | Ctrl+\, Ctrl+M [Global] |
| View.Toolbox | Ctrl+Alt+X [Global] |
| View.ViewCode | Enter [Class Diagram] or F7 [Settings Designer] |
| View.ViewDesigner | Shift+F7 [HTML Editor Source View] |

Window

| COMMANDS | KEYBOARD SHORTCUTS [CONTEXTS] |
|-------------------------------|-------------------------------|
| Window.ActivateDocumentWindow | Esc [Global] |
| Window.CloseDocumentWindow | Ctrl+F4 [Global] |
| Window.NextDocumentWindow | Ctrl+F6 [Global] |
| Window.NextDocumentWindowNav | Ctrl+Tab [Global] |
| Window.NextSplitPane | F6 [Global] |

See also

[Identifying and customizing keyboard shortcuts in Visual Studio](#)

Default Keyboard Shortcuts in Visual Studio

3/21/2018 • 13 min to read • [Edit Online](#)

For more information about keyboard accessibility, see [Accessibility Tips and Tricks](#) and [How to: Use the Keyboard Exclusively](#).

You can access a variety of commands and windows in Visual Studio by choosing the appropriate keyboard shortcut. This topic lists the default shortcuts for the General Development profile, which you might have chosen when you installed Visual Studio. No matter which profile you chose, you can identify the shortcut for a command by opening the **Options** dialog box, expanding the **Environment** node, and then choosing **Keyboard**. You can also customize your shortcuts by assigning a different shortcut to any given command.

For a list of common keyboard shortcuts and other productivity information, see [Default Keyboard Shortcuts for Frequently Used Commands in Visual Studio](#), [Keyboard tips](#), and [Productivity tips](#).

The sections in the following table include commands that are global in that you can access them from anywhere in Visual Studio by using keyboard shortcuts:

| | | | |
|--------------------------|----------------------|------------------------------------|---------------|
| Analyze | Edit | Project | Test |
| Architecture | Editor Context Menus | Project and Solution Context Menus | Test Explorer |
| Build | File | Refactor | Tools |
| Class View Context Menus | Help | Solution Explorer | View |
| Debug | Load Test | Team | Window |
| Debugger Context Menus | Other Context Menus | Team Foundation Context Menus | Azure |
| Diagnostics Hub | | | |

Each section in the following table includes commands for which the keyboard shortcuts are specific to the context for which the section is named.

| | | | |
|------------------------------------|---|-------------------------------------|------------------|
| ADO.NET Entity Data Model Designer | Layer Diagram | Settings Designer | VC Image Editor |
| Class Diagram | Managed Resources Editor | Solution Explorer | VC String Editor |
| Coded UI Test Editor | Merge Editor Window | Team Explorer | View Designer |
| DataSet Editor | Microsoft SQL Server Data Tools, Schema Compare | Team Foundation Build Detail Editor | Visual Studio |

| | | | |
|-------------------------|---|-----------------------|------------------------|
| Difference Viewer | Microsoft SQL Server Data Tools, Table Designer | Test Explorer | Windows Forms Designer |
| DOM Explorer | Microsoft SQL Server Data Tools, T-SQL Editor | Text Editor | Work Item Editor |
| F# Interactive | Microsoft SQL Server Data Tools, T-SQL PDW Editor | UML Activity Diagram | Work Item Query View |
| Graph Document Editor | Page Inspector | UML Class Diagram | Work Item Results View |
| Graphics Diagnostics | Query Designer | UML Component Diagram | Workflow Designer |
| HTML Editor | Query Results | UML Use Case Diagram | XAML UI Designer |
| HTML Editor Design View | Report Designer | VC Accelerator Editor | XML (Text) Editor |
| HTML Editor Source View | Sequence Diagram | VC Dialog Editor | XML Schema Designer |

Global

Analyze

| COMMANDS | KEYBOARD SHORTCUTS |
|--------------------------|--------------------|
| Analyze.NavigateBackward | Shift+Alt+3 |
| Analyze.NavigateForward | Shift+Alt+4 |

Architecture

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------|--------------------|
| Architecture.NewDiagram | Ctrl+\, Ctrl+N |

Build

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------------|--------------------|
| Build.BuildSolution | Ctrl+Shift+B |
| Build.Cancel | Ctrl+Break |
| Build.Compile | Ctrl+F7 |
| Build.RunCodeAnalysisOnSolution | Alt+F11 |

Class View Context Menus

| COMMANDS | KEYBOARD SHORTCUTS |
|---|--------------------|
| ClassViewContextMenus.ClassViewMultiselectProjectreference sItems.Properties | Alt+Enter |

Debug

| COMMANDS | KEYBOARD SHORTCUTS |
|---|--------------------|
| Debug.ApplyCodeChanges | Alt+F10 |
| Debug.Autos | Ctrl+Alt+V, A |
| Debug.BreakAll | Ctrl+Alt+Break |
| Debug.BreakatFunction | Ctrl+B |
| Debug.Breakpoints | Ctrl+Alt+B |
| Debug.CallStack | Ctrl+Alt+C |
| Debug.DeleteAllBreakpoints | Ctrl+Shift+F9 |
| Debug.DiagnosticsHub.Launch | Alt+F2 |
| Debug.Disassembly | Ctrl+Alt+D |
| Debug.DOMExplorer | Ctrl+Alt+V, D |
| Debug.EnableBreakpoint | Ctrl+F9 |
| Debug.Exceptions | Ctrl+Alt+E |
| Debug.GoToPreviousCallorIntelliTraceEvent | Ctrl+Shift+F11 |
| Debug.Graphics.StartDiagnostics | Alt+F5 |
| Debug.Immediate | Ctrl+Alt+I |
| Debug.IntelliTraceCalls | Ctrl+Alt+Y, T |
| Debug.IntelliTraceEvents | Ctrl+Alt+Y, F |
| Debug.JavaScriptConsole | Ctrl+Alt+V, C |
| Debug.Locals | Ctrl+Alt+V, L |
| Debug.LocationToolbar.ProcessCombo | Ctrl+5 |
| Debug.LocationToolbar.StackFrameCombo | Ctrl+7 |
| Debug.LocationToolbar.ThreadCombo | Ctrl+6 |

| COMMANDS | KEYBOARD SHORTCUTS |
|---|------------------------------|
| Debug.LocationToolbar.ToggleCurrentThreadFlaggedState | Ctrl+8 |
| Debug.LocationToolbar.ToggleFlaggedThreads | Ctrl+9 |
| Debug.Memory1 | Ctrl+Alt+M, 1 |
| Debug.Memory2 | Ctrl+Alt+M, 2 |
| Debug.Memory3 | Ctrl+Alt+M, 3 |
| Debug.Memory4 | Ctrl+Alt+M, 4 |
| Debug.Modules | Ctrl+Alt+U |
| Debug.ParallelStacks | Ctrl+Shift+D, S |
| Debug.ParallelWatch1 | Ctrl+Shift+D, 1 |
| Debug.ParallelWatch2 | Ctrl+Shift+D, 2 |
| Debug.ParallelWatch3 | Ctrl+Shift+D, 3 |
| Debug.ParallelWatch4 | Ctrl+Shift+D, 4 |
| Debug.Processes | Ctrl+Alt+Z |
| Debug.QuickWatch | Shift+F9 or Ctrl+Alt+Q |
| Debug.RefreshWindowsapp | Ctrl+Shift+R |
| Debug.Registers | Ctrl+Alt+G |
| Debug.Restart | Ctrl+Shift+F5 |
| Debug.RunWithCursor | Ctrl+F10 |
| Debug.SetNextStatement | Ctrl+Shift+F10 |
| Debug.ShowCallStackonCodeMap | Ctrl+Shift+` |
| Debug.ShowNextStatement | Alt+Num * |
| Debug.Start | F5 |
| Debug.StartWindowsPhoneApplicationAnalysis | Alt+F1 |
| Debug.StartWithoutDebugging | Ctrl+F5 |

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------------|--------------------|
| Debug.StepInto | F11 |
| Debug.StepIntoCurrentProcess | Ctrl+Alt+F11 |
| Debug.StepIntoSpecific | Shift+Alt+F11 |
| Debug.StepOut | Shift+F11 |
| Debug.StepOutCurrentProcess | Ctrl+Shift+Alt+F11 |
| Debug.StepOver | F10 |
| Debug.StepOverCurrentProcess | Ctrl+Alt+F10 |
| Debug.StopDebugging | Shift+F5 |
| Debug.StopPerformanceAnalysis | Shift+Alt+F2 |
| Debug.Tasks | Ctrl+Shift+D, K |
| Debug.Threads | Ctrl+Alt+H |
| Debug.ToggleBreakpoint | F9 |
| Debug.ToggleDisassembly | Ctrl+F11 |
| Debug.Watch1 | Ctrl+Alt+W, 1 |
| Debug.Watch2 | Ctrl+Alt+W, 2 |
| Debug.Watch3 | Ctrl+Alt+W, 3 |
| Debug.Watch4 | Ctrl+Alt+W, 4 |

Debugger Context Menus

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| DebuggerContextMenus.BreakpointsWindow.Delete | Alt+F9, D |
| DebuggerContextMenus.BreakpointsWindow.GoToDisassembly | Alt+F9, A |
| DebuggerContextMenus.BreakpointsWindow.GoToSourceCode | Alt+F9, S |

Diagnostics Hub

| COMMAND | KEYBOARD SHORTCUT |
|-------------------------------|-------------------|
| DiagnosticsHub.StopCollection | Ctrl+Alt+F2 |

Edit

| COMMANDS | |
|---------------------------|--------------------------------------|
| Edit.Copy | Ctrl+C or Ctrl+Ins |
| Edit.Cut | Ctrl+X or Shift+Delete |
| Edit.CycleClipboardRing | Ctrl+Shift+V or Ctrl+Shift+Ins |
| Edit.Delete | Delete |
| Edit.Find | Ctrl+F |
| Edit.FindAllReferences | Shift+F12 |
| Edit.FindinFiles | Ctrl+Shift+F |
| Edit.FindNext | F3 |
| Edit.FindNextSelected | Ctrl+F3 |
| Edit.FindPrevious | Shift+F3 |
| Edit.FindPreviousSelected | Ctrl+Shift+F3 |
| Edit.GenerateMethod | Ctrl+K, Ctrl+M |
| Edit.GoTo | Ctrl+G |
| Edit.GoToDeclaration | Ctrl+F12 |
| Edit.GoToDefinition | F12 |
| Edit.GoToFindCombo | Ctrl+D |
| Edit.GoToNextLocation | F8 |
| Edit.GoToPrevLocation | Shift+F8 |
| Edit.InsertSnippet | Ctrl+K, Ctrl+X |
| Edit.MoveControlDown | Ctrl+Down Arrow |

| COMMANDS | |
|-------------------------------|--|
| Edit.MoveControlDownGrid | Down Arrow |
| Edit.MoveControlLeft | Ctrl+Left Arrow |
| Edit.MoveControlLeftGrid | Left Arrow |
| Edit.MoveControlRight | Ctrl+Right Arrow |
| Edit.MoveControlRightGrid | Right Arrow |
| Edit.MoveControlUp | Ctrl+Up Arrow |
| Edit.MoveControlUpGrid | Up Arrow |
| Edit.NavigateTo | Ctrl+, |
| Edit.NextBookmark | Ctrl+K, Ctrl+N |
| Edit.NextBookmarkInFolder | Ctrl+Shift+K, Ctrl+Shift+N |
| Edit.OpenFile | Ctrl+Shift+G |
| Edit.Paste | <p>Ctrl+V</p> <p>or</p> <p>Shift+Ins</p> |
| Edit.PreviousBookmark | Ctrl+K, Ctrl+P |
| Edit.PreviousBookmarkInFolder | Ctrl+Shift+K, Ctrl+Shift+P |
| Edit.QuickFindSymbol | Shift+Alt+F12 |
| Edit.Redo | <p>Ctrl+Y</p> <p>or</p> <p>Ctrl+Shift+Z</p> <p>or</p> <p>Shift+Alt+Backspace</p> |
| Edit.RefreshRemoteReferences | Ctrl+Shift+J |
| Edit.Replace | Ctrl+H |
| Edit.ReplaceinFiles | Ctrl+Shift+H |
| Edit.SelectAll | Ctrl+A |

| COMMANDS | |
|----------------------------|-------------------------------|
| Edit.SelectNextControl | Tab |
| Edit.SelectPreviousControl | Shift+Tab |
| Edit.ShowTileGrid | Enter |
| Edit.SizeControlDown | Ctrl+Shift+Down Arrow |
| Edit.SizeControlDownGrid | Shift+Down Arrow |
| Edit.SizeControlLeft | Ctrl+Shift+Left Arrow |
| Edit.SizeControlLeftGrid | Shift+Left Arrow |
| Edit.SizeControlRight | Ctrl+Shift+Right Arrow |
| Edit.SizeControlRightGrid | Shift+Right Arrow |
| Edit.SizeControlUp | Ctrl+Shift+Up Arrow |
| Edit.SizeControlUpGrid | Shift+Up Arrow |
| Edit.StopSearch | Alt+F3, S |
| Edit.SurroundWith | Ctrl+K, Ctrl+S |
| Edit.Undo | Ctrl+Z or Alt+Backspace |

Editor Context Menus

| COMMANDS | KEYBOARD SHORTCUTS |
|---|-----------------------------------|
| EditorContextMenus.CodeWindow.Breakpoint.BreakpointEditLabels | Alt+F9, L |
| EditorContextMenus.CodeWindow.CodeMap.ShowItem | Ctrl+` |
| EditorContextMenus.CodeWindow.Execute | Ctrl+Alt+F5 |
| EditorContextMenus.CodeWindow.GoToView | Ctrl+M, Ctrl+G |
| EditorContextMenus.CodeWindow.ToggleHeaderCodeFile | Ctrl+K, Ctrl+O |
| EditorContextMenus.CodeWindow.ViewCallHierarchy | Ctrl+K, Ctrl+T or Ctrl+K, T |

File

| COMMANDS | KEYBOARD SHORTCUTS |
|------------------------|--------------------|
| File.Exit | Alt+F4 |
| File.NewFile | Ctrl+N |
| File.NewProject | Ctrl+Shift+N |
| File.NewWebSite | Shift+Alt+N |
| File.OpenFile | Ctrl+O |
| File.OpenProject | Ctrl+Shift+O |
| File.OpenWebSite | Shift+Alt+O |
| File.Print | Ctrl+P |
| File.SaveAll | Ctrl+Shift+S |
| File.SaveSelectedItems | Ctrl+S |
| File.ViewinBrowser | Ctrl+Shift+W |

Help

| COMMANDS | KEYBOARD SHORTCUTS |
|------------------------------|--------------------|
| Help.AddandRemoveHelpContent | Ctrl+Alt+F1 |
| Help.F1Help | F1 |
| Help.ViewHelp | Ctrl+F1 |
| Help.WindowHelp | Shift+F1 |

Load Test

| COMMAND | KEYBOARD SHORTCUT |
|----------------------------|-------------------|
| LoadTest.JumpToCounterPane | Ctrl+R, Q |

Other Context Menus

| COMMAND | KEYBOARD SHORTCUT |
|--|-------------------|
| OtherContextMenus.MicrosoftDataEntityDesignContext.AddNewDiagram | Insert |

Project

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| Project.AddExistingItem | Shift+Alt+A |
| Project.AddNewItem | Ctrl+Shift+A |
| Project.ClassWizard | Ctrl+Shift+X |
| Project.Override | Ctrl+Alt+Ins |
| Project.Previewchanges | Alt+;, Alt+C |
| Project.Publishselectedfiles | Alt+;, Alt+P |
| Project.Replaceselectedfilesfromserver | Alt+;, Alt+R |

Project and Solution Context Menus

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| ProjectandSolutionContextMenus.Item.MoveDown | Alt+ Down Arrow |
| ProjectandSolutionContextMenus.Item.MoveUp | Alt+ Up Arrow |

Refactor

| COMMANDS | KEYBOARD SHORTCUTS |
|----------------------------|--------------------|
| Refactor.EncapsulateField | Ctrl+R, Ctrl+E |
| Refactor.ExtractInterface | Ctrl+R, Ctrl+I |
| Refactor.ExtractMethod | Ctrl+R, Ctrl+M |
| Refactor.RemoveParameters | Ctrl+R, Ctrl+V |
| Refactor.Rename | Ctrl+R, Ctrl+R |
| Refactor.ReorderParameters | Ctrl+R, Ctrl+O |

Solution Explorer

| COMMANDS | KEYBOARD SHORTCUTS |
|----------------------------------|-----------------------------------|
| SolutionExplorer.OpenFilesFilter | Ctrl+[, O or Ctrl+[, Ctrl+O |

| COMMANDS | KEYBOARD SHORTCUTS |
|---|------------------------------------|
| SolutionExplorer.PendingChangesFilter | Ctrl+[, P or Ctrl+[, Ctrl+P] |
| SolutionExplorer.SyncWithActiveDocument | Ctrl+[, S or Ctrl+[, Ctrl+S] |

Team

| COMMANDS | KEYBOARD SHORTCUTS |
|--------------------------|-----------------------------------|
| Team.Git.GoToGitBranches | Ctrl+0, Ctrl+N or Ctrl+0, N |
| Team.Git.GoToGitChanges | Ctrl+0, Ctrl+G or Ctrl+0, G |
| Team.Git.GoToGitCommits | Ctrl+0, Ctrl+O or Ctrl+0, O |
| Team.TeamExplorerSearch | Ctrl+' |

Team Foundation Context Menus

| COMMANDS | KEYBOARD SHORTCUTS |
|---|-----------------------------------|
| TeamFoundationContextMenus.Commands.GoToBuilds | Ctrl+0, Ctrl+B or Ctrl+0, B |
| TeamFoundationContextMenus.Commands.GoToConnect | Ctrl+0, Ctrl+C or Ctrl+0, C |

| COMMANDS | KEYBOARD SHORTCUTS |
|--|-----------------------------------|
| TeamFoundationContextMenus.Commands.GoToDocuments | Ctrl+0, Ctrl+D or Ctrl+0, D |
| TeamFoundationContextMenus.Commands.GoToHome | Ctrl+0, Ctrl+H or Ctrl+0, H |
| TeamFoundationContextMenus.Commands.GoToMyWork | Ctrl+0, Ctrl+M or Ctrl+0, M |
| TeamFoundationContextMenus.Commands.GoToPendingChanges | Ctrl+0, Ctrl+P or Ctrl+0, P |
| TeamFoundationContextMenus.Commands.GoToReports | Ctrl+0, Ctrl+R or Ctrl+0, R |
| TeamFoundationContextMenus.Commands.GoToSettings | Ctrl+0, Ctrl+S or Ctrl+0, S |
| TeamFoundationContextMenus.Commands.GoToWebAccess | Ctrl+0, Ctrl+A or Ctrl+0, A |
| TeamFoundationContextMenus.Commands.GoToWorkItems | Ctrl+0, Ctrl+W or Ctrl+0, W |

Test

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------------|--------------------|
| Test.UseCodedUITestBuilder | Ctrl+\, Ctrl+C |
| Test.UseExistingActionRecording | Ctrl+\, Ctrl+A |

Test Explorer

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------------------|--------------------|
| TestExplorer.DebugAllTests | Ctrl+R, Ctrl+A |
| TestExplorer.DebugAllTestsInContext | Ctrl+R, Ctrl+T |
| TestExplorer.RepeatLastRun | Ctrl+R, L |
| TestExplorer.RunAllTests | Ctrl+R, A |
| TestExplorer.RunAllTestsInContext | Ctrl+R, T |

Tools

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------|--|
| Tools.AttachtoProcess | Ctrl+Alt+P |
| Tools.CodeSnippetsManager | Ctrl+K, Ctrl+B |
| Tools.ForceGC | Ctrl+Shift+Alt+F12, Ctrl+Shift+Alt+F12 |
| Tools.GoToCommandLine | Ctrl+ / |

View

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------------|-----------------------------------|
| View.AllWindows | Shift+Alt+M |
| View.ArchitectureExplorer | Ctrl+\, Ctrl+R |
| View.Backward | Alt+Left Arrow |
| View.BookmarkWindow | Ctrl+K, Ctrl+W |
| View.BrowseNext | Ctrl+Shift+1 |
| View.BrowsePrevious | Ctrl+Shift+2 |
| View.CallHierarchy | Ctrl+Alt+K |
| View.ClassView | Ctrl+Shift+C |
| View.ClassViewGoToSearchCombo | Ctrl+K, Ctrl+V |
| View.CodeDefinitionWindow | Ctrl+\, D or Ctrl+\, Ctrl+D |
| View.CommandWindow | Ctrl+Alt+A |

| COMMANDS | KEYBOARD SHORTCUTS |
|-----------------------------------|--|
| View.DataSources | Shift+Alt+D |
| View.DocumentOutline | Ctrl+Alt+T |
| View.EditLabel | F2 |
| View.ErrorList | <p>Ctrl+_ E</p> <p>or</p> <p>Ctrl+_ Ctrl+E</p> |
| View.F#Interactive | Ctrl+Alt+F |
| View.FindSymbolResults | Ctrl+Alt+F12 |
| View.Forward | Alt+ Right Arrow |
| View.ForwardBrowseContext | Ctrl+Shift+7 |
| View.FullScreen | Shift+Alt+Enter |
| View.NavigateBackward | Ctrl+- |
| View.NavigateForward | Ctrl+Shift+- |
| View.NextError | Ctrl+Shift+F12 |
| View.Notifications | <p>Ctrl+W, N</p> <p>or</p> <p>Ctrl+W, Ctrl+N</p> |
| View.ObjectBrowser | Ctrl+Alt+J |
| View.ObjectBrowserGoToSearchCombo | Ctrl+K, Ctrl+R |
| View.Output | Ctrl+Alt+O |
| View.PopBrowseContex | Ctrl+Shift+8 |
| View.PropertiesWindow | F4 |
| View.PropertyPages | Shift+F4 |
| View.ResourceView | Ctrl+Shift+E |
| View.ServerExplorer | Ctrl+Alt+S |

| COMMANDS | KEYBOARD SHORTCUTS |
|------------------------------|-----------------------------------|
| View.ShowSmartTag | Shift+Alt+F10 or Ctrl+. |
| View.SolutionExplorer | Ctrl+Alt+L |
| View.SQLServerObjectExplorer | Ctrl+_ Ctrl+S |
| View.TaskList | Ctrl+_ T or Ctrl+_ Ctrl+T |
| View.TfsTeamExplorer | Ctrl+_ Ctrl+M |
| View.Toolbox | Ctrl+Alt+X |
| View.UMLModelExplorer | Ctrl+_ Ctrl+U |
| View.ViewCode | F7 |
| View.ViewDesigner | Shift+F7 |
| View.WebBrowser | Ctrl+Alt+R |
| View.ZoomIn | Ctrl+Shift+. |
| View.ZoomOut | Ctrl+Shift+, |

Window

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------------|----------------------|
| Window.ActivateDocumentWindow | Esc |
| Window.AddTabtoSelection | Ctrl+Shift+Alt+Space |
| Window.CloseDocumentWindow | Ctrl+F4 |
| Window.CloseToolWindow | Shift+Esc |
| Window.KeepTabOpen | Ctrl+Alt+Home |
| Window.MovetoNavigationBar | Ctrl+F2 |
| Window.NextDocumentWindow | Ctrl+F6 |
| Window.NextDocumentWindowNav | Ctrl+Tab |

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------------------|----------------------------------|
| Window.NextPane | Alt+F6 |
| Window.NextSplitPane | F6 |
| Window.NextTab | Ctrl+Alt+PgDn or Ctrl+PgDn |
| Window.NextTabandAddtoSelection | Ctrl+Shift+Alt+PgDn |
| Window.NextToolWindowNav | Alt+F7 |
| Window.PreviousDocumentWindow | Ctrl+Shift+F6 |
| Window.PreviousDocumentWindowNav | Ctrl+Shift+Tab |
| Window.PreviousPane | Shift+Alt+F6 |
| Window.PreviousSplitPane | Shift+F6 |
| Window.PreviousTab | Ctrl+Alt+PgUp or Ctrl+PgUp |
| Window.PreviousTabandAddtoSelection | Ctrl+Shift+Alt+PgUp |
| Window.PreviousToolWindowNav | Shift+Alt+F7 |
| Window.QuickLaunch | Ctrl+Q |
| Window.QuickLaunchPreviousCategory | Ctrl+Shift+Q |
| Window.ShowDockMenu | Alt+- |
| Window.ShowEzMDIFileList | Ctrl+Alt+Down Arrow |
| Window.SolutionExplorerSearch | Ctrl+; |
| Window.WindowSearch | Alt+` |

Azure

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| WindowsAzure.RetryMobileServiceScriptOperation | Ctrl+Num *, Ctrl+R |
| WindowsAzure.ShowMobileServiceScriptErrorDetails | Ctrl+Num *, Ctrl+D |

ADO.NET Entity Data Model Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Down | Alt+ Down Arrow |
| OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Down5 | Alt+ PgDn |
| OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.ToBottom | Alt+ End |
| OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.ToTop | Alt+ Home |
| OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Up | Alt+ Up Arrow |
| OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Up5 | Alt+ PgUp |
| OtherContextMenus.MicrosoftDataEntityDesignContext.Refactor.Rename | Ctrl+R, R |
| OtherContextMenus.MicrosoftDataEntityDesignContext.RemovefromDiagram | Shift+ Del |
| View.EntityDataModelBrowser | Ctrl+1 |
| View.EntityDataModelMappingDetails | Ctrl+2 |

Class Diagram

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------------|--------------------|
| ClassDiagram.Collapse | Num - |
| ClassDiagram.Expand | Num + |
| Edit.Delete | Ctrl+ Del |
| Edit.ExpandCollapseBaseTypeList | Shift+ Alt+ B |
| Edit.NavigateToLollipop | Shift+ Alt+ L |
| Edit.RemovefromDiagram | Delete |
| View.ViewCode | Enter |

Coded UI Test Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| OtherContextMenus.UITestEditorContextMenu.CopyReferenceClipboard | Ctrl+C |
| OtherContextMenus.UITestEditorContextMenu.InsertDelayBefore | Ctrl+Alt+D |
| OtherContextMenus.UITestEditorContextMenu.LocateAll | Shift+Alt+L |
| OtherContextMenus.UITestEditorContextMenu.LocatetheUIControl | Ctrl+Shift+L |
| OtherContextMenus.UITestEditorContextMenu.Movecode | Ctrl+Alt+C |
| OtherContextMenus.UITestEditorContextMenu.Splitintoanewmethod | Ctrl+Shift+T |

DataSet Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| OtherContextMenus.ColumnContext.InsertColumn | Insert |
| OtherContextMenus.DbTableContext.Add.Column | Ctrl+L |

Difference Viewer

| Commands | Keyboard Shortcuts |
|--|-------------------------|
| Diff.IgnoreTrimWhitespace | Ctrl+_ Ctrl+Spacebar |
| Diff.InlineView | Ctrl+_ Ctrl+1 |
| Diff.LeftOnlyView | Ctrl+_ Ctrl+3 |
| Diff.NextDifference | F8 |
| Diff.PreviousDifference | Shift+F8 |
| Diff.RightOnlyView | Ctrl+_ Ctrl+4 |
| Diff.SideBySideView | Ctrl+_ Ctrl+2 |
| Diff.SwitchBetweenLeftAndRight | Ctrl+_ Ctrl+Tab |
| Diff.SynchronizeViewToggle | Ctrl+_ Ctrl+Down Arrow |
| EditorContextMenus.CodeWindow.AddComment | Ctrl+Shift+K |

| | |
|---|--------------|
| EditorContextMenus.CodeWindow.EditLocalFile | Ctrl+Shift+P |
|---|--------------|

DOM Explorer

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------|--------------------|
| DOMExplorer.Refresh | F5 |
| DOMExplorer.SelectElement | Ctrl+B |
| DOMExplorer.ShowLayout | Ctrl+Shift+I |

F# Interactive

| COMMAND | KEYBOARD SHORTCUT |
|---|-------------------|
| OtherContextMenus.FSIConsoleContext.CancelInteractiveEvaluation | Ctrl+Break |

Graph Document Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|--|---------------------------------|
| ArchitectureContextMenus.DirectedGraphContextMenu.Advanced.Add.AddNode | Insert |
| ArchitectureContextMenus.DirectedGraphContextMenu.Advanced.Select.BothDependencies | B |
| ArchitectureContextMenus.DirectedGraphContextMenu.Advanced.Select.IncomingDependencies | I |
| ArchitectureContextMenus.DirectedGraphContextMenu.Advanced.Select.OutgoingDependencies | O |
| ArchitectureContextMenus.DirectedGraphContextMenu.New Comment | Ctrl+Shift+K or Ctrl+E, C |
| ArchitectureContextMenus.DirectedGraphContextMenu.Remove | Delete |
| ArchitectureContextMenus.DirectedGraphContextMenu.Rename | F2 |

Graphics Diagnostics

| COMMANDS | KEYBOARD SHORTCUTS |
|----------------------------------|------------------------|
| Debug.Graphics.CaptureFrame | None |
| Graphics.MovePixelSelectionDown | Shift+Alt+ Down Arrow |
| Graphics.MovePixelSelectionLeft | Shift+Alt+ Left Arrow |
| Graphics.MovePixelSelectionRight | Shift+Alt+ Right Arrow |
| Graphics.MovePixelSelectionUp | Shift+Alt+ Up Arrow |
| Graphics.ZoomToActualSize | Shift+Alt+0 |
| Graphics.ZoomToFitInWindow | Shift+Alt+9 |
| Graphics.ZoomIn | Shift+Alt+= |
| Graphics.ZoomOut | Shift+Alt+- |

HTML Editor

| COMMAND | KEYBOARD SHORTCUT |
|---|-------------------|
| OtherContextMenu.HTMLContext.GoToController | Ctrl+M, Ctrl+G |

HTML Editor Design View

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------|-----------------------|
| Edit.MoveControlDown | Ctrl+ Down Arrow |
| Edit.MoveControlUp | Ctrl+ Up Arrow |
| Format.Bold | Ctrl+B |
| Format.ConverttoHyperlink | Ctrl+L |
| Format.InsertBookmark | Ctrl+Shift+L |
| Format.Italic | Ctrl+I |
| Format.Underline | Ctrl+U |
| Project.AddContentPage | Ctrl+M, Ctrl+C |
| Table.ColumntotheLeft | Ctrl+Alt+ Left Arrow |
| Table.ColumntotheRight | Ctrl+Alt+ Right Arrow |
| Table.RowAbove | Ctrl+Alt+ Up Arrow |

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------------|---------------------|
| Table.RowBelow | Ctrl+Alt+Down Arrow |
| View.ASP.NETNonvisualControls | Ctrl+Shift+N |
| View.EditMaster | Ctrl+M, Ctrl+M |
| View.NextView | Ctrl+PgDn |
| View.ShowSmartTag | Shift+Alt+F10 |
| View.ViewMarkup | Shift+F7 |
| Window.PreviousTab | Ctrl+PgUp |

HTML Editor Source View

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| OtherContextMenu.HTMLEContext.GoToController | Ctrl+M, Ctrl+G |
| View.NextView | Ctrl+PgDn |
| View.SynchronizeViews | Ctrl+Shift+Y |
| View.ViewDesigner | Shift+F7 |
| Window.PreviousTab | Ctrl+PgUp |

Layer Diagram

| COMMAND | KEYBOARD SHORTCUT |
|-------------|-------------------|
| Edit.Delete | Shift+Delete |

Managed Resources Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|----------------------|--------------------|
| Edit.EditCell | F2 |
| Edit.Remove | Delete |
| Edit.RemoveRow | Ctrl+Delete |
| Edit.SelectionCancel | Escape |
| Resources.Audio | Ctrl+4 |

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------|--------------------|
| Resources.Files | Ctrl+5 |
| Resources.Icons | Ctrl+3 |
| Resources.Images | Ctrl+2 |
| Resources.Other | Ctrl+6 |
| Resources.Strings | Ctrl+1 |

Merge Editor Window

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| TeamFoundationContextMenus.MergeContextMenu.SetFocusOnLeftWindow | Alt+1 |
| TeamFoundationContextMenus.MergeContextMenu.SetFocusOnResultWindow | Alt+2 |
| TeamFoundationContextMenus.MergeContextMenu.SetFocusOnRightWindow | Alt+3 |

Microsoft SQL Server Data Tools, Schema Compare

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------------------|--------------------|
| SQL.SSDTSchemaCompareCompare | Shift+Alt+C |
| SQL.SSDTSchemaCompareGenerateScript | Shift+Alt+G |
| SQL.SSDTSchemaCompareNextChange | Shift+Alt+. |
| SQL.SSDTSchemaComparePreviousChange | Shift+Alt+, |
| SQL.SSDTSchemaCompareStop | Alt+Break |
| SQL.SSDTSchemaCompareWriteUpdates | Shift+Alt+U |

Microsoft SQL Server Data Tools, Table Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|----------------|--------------------|
| CommitAllEdits | Shift+Alt+U |

| COMMANDS | KEYBOARD SHORTCUTS |
|-----------------------|---|
| SQL.ExpandWildcards | Ctrl+R, E or Ctrl+R, Ctrl+E |
| SQL.FullyqualifyNames | Ctrl+R, Q or Ctrl+R, Ctrl+Q |
| SQL.MovetoSchema | Ctrl+R, M or Ctrl+R, Ctrl+M |
| SQL.Rename | F2 or Ctrl+R, R or Ctrl+R, Ctrl+R |
| ViewFileInScriptPanel | Shift+Alt+PgDn |

Microsoft SQL Server Data Tools, T-SQL Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------|-----------------------------------|
| CommitAllEdits | Shift+Alt+U |
| SQL.ExecuteWithDebugger | Alt+F5 |
| SQL.ExpandWildcards | Ctrl+R, E or Ctrl+R, Ctrl+E |
| SQL.FullyqualifyNames | Ctrl+R, Q or Ctrl+R, Ctrl+Q |
| SQL.MovetoSchema | Ctrl+R, M or Ctrl+R, Ctrl+M |

| COMMANDS | KEYBOARD SHORTCUTS |
|-----------------------------------|---|
| SQL.Rename | F2 or Ctrl+R, R or Ctrl+R, Ctrl+R |
| SQL.TSqlEditorCancelQuery | Alt+ Break |
| SQL.TSqlEditorExecuteQuery | Ctrl+Shift+E |
| SQL.TSqlEditorResultsAsFile | Ctrl+D, F |
| SQL.TSqlEditorResultsAsGrid | Ctrl+D, G |
| SQL.TSqlEditorResultsAsText | Ctrl+D, T |
| SQL.TSqlEditorShowEstimatedPlan | Ctrl+D, E |
| SQL.TSqlEditorToggleExecutionPlan | Ctrl+D, A |
| SQL.TSqlEditorToggleResultsPane | Ctrl+D, R |
| TSqlEditorCloneQuery | Ctrl+Alt+N |
| TSqlEditorDatabaseCombo | Shift+Alt+PgDn |

Microsoft SQL Server Data Tools, T-SQL PDW Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|-----------------------------------|--------------------|
| SQL.TSqlEditorCancelQuery | Alt+ Break |
| SQL.TSqlEditorExecuteQuery | Ctrl+Shift+E |
| SQL.TSqlEditorResultsAsFile | Ctrl+D, F |
| SQL.TSqlEditorResultsAsGrid | Ctrl+D, G |
| SQL.TSqlEditorResultsAsText | Ctrl+D, T |
| SQL.TSqlEditorShowEstimatedPlan | Ctrl+D, E |
| SQL.TSqlEditorToggleExecutionPlan | Ctrl+D, A |
| SQL.TSqlEditorToggleResultsPane | Ctrl+D, R |
| TSqlEditorCloneQuery | Ctrl+Alt+N |

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------|--------------------|
| TSqlEditorDatabaseCombo | Shift+Alt+PgDn |

Page Inspector

| COMMAND | KEYBOARD SHORTCUT |
|------------------------|-------------------|
| PageInspector.Minimize | F12 |

Query Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|------------------------------------|--------------------|
| QueryDesigner.CancelRetrievingData | Ctrl+T |
| QueryDesigner.Criteria | Ctrl+2 |
| QueryDesigner.Diagram | Ctrl+1 |
| QueryDesigner.ExecuteSQL | Ctrl+R |
| QueryDesigner.GotoRow | Ctrl+G |
| QueryDesigner.JoinMode | Ctrl+Shift+J |
| QueryDesigner.Results | Ctrl+4 |
| QueryDesigner.SQL | Ctrl+3 |

Query Results

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------|--------------------|
| SQL.QueryResultsNewRow | Alt+End |
| SQL.QueryResultsRefresh | Shift+Alt+R |
| SQL.QueryResultsStop | Alt+Break |

Report Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------|--------------------|
| Edit.BreakLine | Enter |
| Edit.CharLeft | Left Arrow |
| Edit.CharLeftExtend | Shift+Left Arrow |

| COMMANDS | KEYBOARD SHORTCUTS |
|-----------------------|------------------------|
| Edit.CharRight | Right Arrow |
| Edit.CharRightExtend | Shift+Right Arrow |
| Edit.InsertTab | Tab |
| Edit.LineDown | Down Arrow |
| Edit.LineDownExtend | Shift+Down Arrow |
| Edit.LineUp | Up Arrow |
| Edit.LineUpExtend | Shift+Up Arrow |
| Edit.MoveControlDown | Ctrl+Down Arrow |
| Edit.MoveControlLeft | Ctrl+Left Arrow |
| Edit.MoveControlRight | Ctrl+Right Arrow |
| Edit.MoveControlUp | Ctrl+Up Arrow |
| Edit.SelectionCancel | Esc |
| Edit.SizeControlDown | Ctrl+Shift+Down Arrow |
| Edit.SizeControlLeft | Ctrl+Shift+Left Arrow |
| Edit.SizeControlRight | Ctrl+Shift+Right Arrow |
| Edit.SizeControlUp | Ctrl+Shift+Up Arrow |
| Edit.TabLeft | Shift+Tab |
| View.ReportData | Ctrl+Alt+D |

Sequence Diagram

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| ArchitectureDesigner.Sequence.NavigateToCode | F12 |
| Edit.Delete | Shift+Del |

Settings Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------|--------------------|
| Edit.EditCell | F2 |

| COMMANDS | KEYBOARD SHORTCUTS |
|----------------------|--------------------|
| Edit.RemoveRow | Ctrl+Delete |
| Edit.SelectionCancel | Esc |
| View.ViewCode | F7 |

Solution Explorer

| COMMAND | KEYBOARD SHORTCUT |
|---|-------------------|
| ClassViewContextMenus.ClassViewProject.View.ViewInPageInspector | Ctrl+K, Ctrl+G |

Team Explorer

| COMMAND | KEYBOARD SHORTCUT |
|--|-------------------|
| Edit.Delete | Delete |
| File.Rename | F2 |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerNavigation | Alt+Home |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerNextSectionContent | Alt+Down Arrow |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerPageContent | Alt+0 |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerPreviousSectionContent | Alt+Up Arrow |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerSection1Content | Alt+1 |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerSection2Content | Alt+2 |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerSection3Content | Alt+3 |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerSection4Content | Alt+4 |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerSection5Content | Alt+5 |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerSection6Content | Alt+6 |

| COMMAND | KEYBOARD SHORTCUT |
|--|-------------------|
| TeamFoundationContextMenus.Commands.GoToTeamExplorerSection7Content | Alt+7 |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerSection8Content | Alt+8 |
| TeamFoundationContextMenus.Commands.GoToTeamExplorerSection9Content | Alt+9 |
| TeamFoundationContextMenus.Commands.TeamExplorerNavigateBackward | Alt+ Left Arrow |
| TeamFoundationContextMenus.Commands.TeamExplorerNavigateForward | Alt+ Right Arrow |
| TeamFoundationContextMenus.MyWorkPageInProgress.TfsContextMyWorkPageCreateCopyWI | Shift+Alt+C |
| TeamFoundationContextMenus.MyWorkPageInProgress.TfsContextMyWorkPageNewLinkedWI | Shift+Alt+L |
| View.Refresh | F5 |

Team Foundation Build Detail Editor

| COMMAND | KEYBOARD SHORTCUT |
|--------------|-------------------|
| View.Refresh | F5 |

Test Explorer

| COMMAND | KEYBOARD SHORTCUT |
|-----------------------|-------------------|
| TestExplorer.OpenTest | F12 |

Text Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------|----------------------------|
| Edit.BreakLine | Enter or Shift+Enter |
| Edit.CharLeft | Left Arrow |
| Edit.CharLeftExtend | Shift+Left Arrow |
| Edit.CharLeftExtendColumn | Shift+Alt+Left Arrow |

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------------|-------------------------------------|
| Edit.CharRight | Right Arrow |
| Edit.CharRightExtend | Shift+Right Arrow |
| Edit.CharRightExtendColumn | Shift+Alt+Right Arrow |
| Edit.CharTranspose | Ctrl+T |
| Edit.ClearBookmarks | Ctrl+K, Ctrl+L |
| Edit.CollapseAllOutlining | Ctrl+M, Ctrl+A |
| Edit.CollapseCurrentRegion | Ctrl+M, Ctrl+S |
| Edit.CollapseTag | Ctrl+M, Ctrl+T |
| Edit.CollapseToDefinitions | Ctrl+M, Ctrl+O |
| Edit.CommentSelection | Ctrl+K, Ctrl+C |
| Edit.CompleteWord | Ctrl+Space or Alt+Right Arrow |
| Edit.CopyParameterTip | Ctrl+Shift+Alt+C |
| Edit.DecreaseFilterLevel | Alt+, |
| Edit.DeleteBackwards | Backspace or Shift+Bkspce |
| Edit.DeleteHorizontalWhiteSpace | Ctrl+K, Ctrl+\ |
| Edit.DocumentEnd | Ctrl+End |
| Edit.DocumentEndExtend | Ctrl+Shift+End |
| Edit.DocumentStart | Ctrl+Home |
| Edit.DocumentStartExtend | Ctrl+Shift+Home |
| Edit.ExpandAllOutlining | Ctrl+M, Ctrl+X |
| Edit.ExpandCurrentRegion | Ctrl+M, Ctrl+E |
| Edit.FormatDocument | Ctrl+K, Ctrl+D |

| COMMANDS | KEYBOARD SHORTCUTS |
|----------------------------|----------------------|
| Edit.FormatSelection | Ctrl+K, Ctrl+F |
| Edit.GotoBrace | Ctrl+] |
| Edit.GotoBraceExtend | Ctrl+Shift+] |
| Edit.HideSelection | Ctrl+M, Ctrl+H |
| Edit.IncreaseFilterLevel | Alt+. . |
| Edit.IncrementalSearch | Ctrl+I |
| Edit.InsertTab | Tab |
| Edit.LineCut | Ctrl+L |
| Edit.LineDelete | Ctrl+Shift+L |
| Edit.LineDown | Down Arrow |
| Edit.LineDownExtend | Shift+Down Arrow |
| Edit.LineDownExtendColumn | Shift+Alt+Down Arrow |
| Edit.LineEnd | End |
| Edit.LineEndExtend | Shift+End |
| Edit.LineEndExtendColumn | Shift+Alt+End |
| Edit.LineOpenAbove | Ctrl+Enter |
| Edit.LineOpenBelow | Ctrl+Shift+Enter |
| Edit.LineStart | Home |
| Edit.LineStartExtend | Shift+Home |
| Edit.LineStartExtendColumn | Shift+Alt+Home |
| Edit.LineTranspose | Shift+Alt+T |
| Edit.LineUp | Up Arrow |
| Edit.LineUpExtend | Shift+Up Arrow |
| Edit.LineUpExtendColumn | Shift+Alt+Up Arrow |
| Edit.ListMembers | Ctrl+J |

| COMMANDS | KEYBOARD SHORTCUTS |
|-----------------------------------|-----------------------|
| Edit.MakeLowercase | Ctrl+U |
| Edit.MakeUppercase | Ctrl+Shift+U |
| Edit.MoveSelectedLinesDown | Alt+ Down Arrow |
| Edit.MoveSelectedLinesUp | Alt+ Up Arrow |
| Edit.NextHighlightedReference | Ctrl+Shift+Down Arrow |
| Edit.OvertypeMode | Insert |
| Edit.PageDown | PgDn |
| Edit.PageDownExtend | Shift+PgDn |
| Edit.PageUp | PgUp |
| Edit.PageUpExtend | Shift+PgUp |
| Edit.ParameterInfo | Ctrl+Shift+Spacebar |
| Edit.PasteParameterTip | Ctrl+Shift+Alt+P |
| Edit.PeekBackward | Ctrl+Alt+- |
| Edit.PeekDefinition | Alt+F12 |
| Edit.PeekForward | Ctrl+Alt+= |
| Edit.PreviousHighlightedReference | Ctrl+Shift+Up Arrow |
| Edit.QuickInfo | Ctrl+K, Ctrl+I |
| Edit.ReverseIncrementalSearch | Ctrl+Shift+I |
| Edit.ScrollLineDown | Ctrl+ Down Arrow |
| Edit.ScrollLineUp | Ctrl+ Up Arrow |
| Edit.SelectCurrentWord | Ctrl+W |
| Edit.SelectionCancel | Escape |
| Edit.SelectToLastGoBack | Ctrl+= |
| Edit.ShowCodeLensMenu | Alt+` |
| Edit.StopHidingCurrent | Ctrl+M, Ctrl+U |

| COMMANDS | KEYBOARD SHORTCUTS |
|---|----------------------------|
| Edit.StopOutlining | Ctrl+M, Ctrl+P |
| Edit.SwapAnchor | Ctrl+K, Ctrl+A |
| Edit.TabLeft | Shift+Tab |
| Edit.ToggleAllOutlining | Ctrl+M, Ctrl+L |
| Edit.ToggleBookmark | Ctrl+K, Ctrl+K |
| Edit.ToggleCompletionMode | Ctrl+Alt+Space |
| Edit.ToggleOutliningExpansion | Ctrl+M, Ctrl+M |
| Edit.ToggleTaskListShortcut | Ctrl+K, Ctrl+H |
| Edit.ToggleWordWrap | Ctrl+E, Ctrl+W |
| Edit.UncommentSelection | Ctrl+K, Ctrl+U |
| Edit.ViewBottom | Ctrl+PgDn |
| Edit.ViewBottomExtend | Ctrl+Shift+PgDn |
| Edit.ViewTop | Ctrl+PgUp |
| Edit.ViewTopExtend | Ctrl+Shift+PgUp |
| Edit.ViewWhiteSpace | Ctrl+R, Ctrl+W |
| Edit.WordDeleteToEnd | Ctrl+Delete |
| Edit.WordDeleteToStart | Ctrl+Backspace |
| Edit.WordNext | Ctrl+Right Arrow |
| Edit.WordNextExtend | Ctrl+Shift+Right Arrow |
| Edit.WordNextExtendColumn | Ctrl+Shift+Alt+Right Arrow |
| Edit.WordPrevious | Ctrl+Left Arrow |
| Edit.WordPreviousExtend | Ctrl+Shift+Left Arrow |
| Edit.WordPreviousExtendColumn | Ctrl+Shift+Alt+Left Arrow |
| Edit.WordTranspose | Ctrl+Shift+T |
| EditorContextMenu.CodeWindow.ExecuteInInteractive | Alt+Enter |

| COMMANDS | KEYBOARD SHORTCUTS |
|--|--------------------|
| EditorContextMenus.CodeWindow.ExecuteLineInInteractive | Alt+' |
| OtherContextMenus.HTMLContext.ViewinPageInspector | Ctrl+K, Ctrl+G |
| TeamFoundationContextMenu.Annotate.TfsAnnotateMoveNextRegion | Alt+ PgDn |
| TeamFoundationContextMenu.Annotate.TfsAnnotateMovePreviousRegion | Alt+ PgUp |

UML Activity Diagram

| COMMAND | KEYBOARD SHORCTUT |
|-------------|-------------------|
| Edit.Delete | Shift+ Del |

UML Class Diagram

| COMMAND | KEYBOARD SHORCTUT |
|----------------------|-------------------|
| Edit.DeleteFromModel | Shift+ Del |

UML Component Diagram

| COMMAND | KEYBOARD SHORCTUT |
|----------------------|-------------------|
| Edit.DeleteFromModel | Shift+ Del |

UML Use Case Diagram

| COMMAND | KEYBOARD SHORCTUT |
|----------------------|-------------------|
| Edit.DeleteFromModel | Shift+ Del |

VC Accelerator Editor

| COMMANDS | KEYBOARD SHORCTUTS |
|---------------------|--------------------|
| Edit.NewAccelerator | Insert |
| Edit.NextKeyTyped | Ctrl+W |

VC Dialog Editor

| COMMANDS | KEYBOARD SHORCTUTS |
|----------------------|--------------------|
| Edit.MoveControlDown | Down Arrow |

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------|---|
| Edit.MoveControlLeft | Left Arrow |
| Edit.MoveControlRight | Right Arrow |
| Edit.MoveControlUp | Up Arrow |
| Edit.ScrollColumnLeft | Ctrl+Left Arrow |
| Edit.ScrollColumnRight | Ctrl+Right Arrow |
| Edit.ScrollLineDown | Ctrl+Down Arrow |
| Edit.ScrollLineUp | Ctrl+Up Arrow |
| Edit.SizeControlDown | Shift+Down Arrow |
| Edit.SizeControlLeft | Shift+Left Arrow |
| Edit.SizeControlRight | Shift+Right Arrow |
| Edit.SizeControlUp | Shift+Up Arrow |
| Format.AlignBottoms | Ctrl+Shift+Down Arrow |
| Format.AlignCenters | Shift+F9 |
| Format.AlignLefts | Ctrl+Shift+Left Arrow |
| Format.AlignMiddles | F9 |
| Format.AlignRights | Ctrl+Shift+Right Arrow |
| Format.AlignTops | Ctrl+Shift+Up Arrow |
| Format.ButtonBottom | Ctrl+B |
| Format.ButtonRight | Ctrl+R |
| Format.CenterHorizontal | Ctrl+Shift+F9 |
| Format.CenterVertical | Ctrl+F9 |
| Format.CheckMnemonics | Ctrl+M |
| Format.SizetoContent | Shift+F7 |
| Format.SpaceAcross | Alt+Right Arrow or Alt+Left Arrow |

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------|--------------------------------------|
| Format.SpaceDown | Alt+Up Arrow or Alt+Down Arrow |
| Format.TabOrder | Ctrl+D |
| Format.TestDialog | Ctrl+T |
| Format.ToggleGuides | Ctrl+G |

VC Image Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|----------------------------------|--------------------|
| Image.AirbrushTool | Ctrl+A |
| Image.BrushTool | Ctrl+B |
| Image.CopyandOutlineSelection | Ctrl+Shift+U |
| Image.DrawOpaque | Ctrl+J |
| Image.EllipseTool | Alt+P |
| Image.EraseTool | Ctrl+Shift+I |
| Image.FilledEllipseTool | Ctrl+Shift+Alt+P |
| Image.FilledRectangleTool | Ctrl+Shift+Alt+R |
| Image.FilledRoundedRectangleTool | Ctrl+Shift+Alt+W |
| Image.FillTool | Ctrl+F |
| Image.FlipHorizontal | Ctrl+H |
| Image.FlipVertical | Shift+Alt+H |
| Image.LargerBrush | Ctrl+= |
| Image.LineTool | Ctrl+L |
| Image.MagnificationTool | Ctrl+M |
| Image.Magnify | Ctrl+Shift+M |
| Image.NewImageType | Insert |

| COMMANDS | KEYBOARD SHORTCUTS |
|------------------------------------|---|
| Image.NextColor | Ctrl+] or Ctrl+Right Arrow |
| Image.NextRightColor | Ctrl+Shift+] |
| | or |
| | Ctrl+Shift+Right Arrow |
| Image.OutlinedEllipseTool | Shift+Alt+P |
| Image.OutlinedRectangleTool | Shift+Alt+R |
| Image.OutlinedRoundedRectangleTool | Shift+Alt+W |
| Image.PencilTool | Ctrl+I |
| Image.PreviousColor | Ctrl+[or Ctrl+Left Arrow |
| Image.PreviousRightColor | Ctrl+Shift+[or Ctrl+Shift+Left Arrow |
| Image.RectangleSelectionTool | Shift+Alt+S |
| Image.RectangleTool | Alt+R |
| Image.Rotate90Degrees | Ctrl+Shift+H |
| Image.RoundedRectangleTool | Alt+W |
| Image.ShowGrid | Ctrl+Alt+S |
| Image.ShowTileGrid | Ctrl+Shift+Alt+S |
| Image.SmallBrush | Ctrl+. |
| Image.SmallerBrush | Ctrl+- |
| Image.TextTool | Ctrl+T |
| Image.UseSelectionasBrush | Ctrl+U |

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------|---------------------------------------|
| Image.ZoomIn | Ctrl+Shift+. or Ctrl+Up Arrow |
| Image.ZoomOut | Ctrl+Shift+, or Ctrl+Down Arrow |

VC String Editor

| COMMAND | KEYBOARD SHORTCUT |
|----------------|-------------------|
| Edit.NewString | Insert |

View Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|------------------------------------|--------------------|
| QueryDesigner.CancelRetrievingData | Ctrl+T |
| QueryDesigner.Criteria | Ctrl+2 |
| QueryDesigner.Diagram | Ctrl+1 |
| QueryDesigner.ExecuteSQL | Ctrl+R |
| QueryDesigner.GotoRow | Ctrl+G |
| QueryDesigner.JoinMode | Ctrl+Shift+J |
| QueryDesigner.Results | Ctrl+4 |
| QueryDesigner.SQL | Ctrl+3 |

Visual Studio

| COMMAND | KEYBOARD SHORTCUT |
|---|-------------------|
| OtherContextMenus.ORDesignerContext.HideMethodsPane | Ctrl+1 |

Windows Forms Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|----------------|--------------------|
| Edit.BreakLine | Enter |

| COMMANDS | KEYBOARD SHORTCUTS |
|--------------------------|------------------------|
| Edit.CharLeft | Left Arrow |
| Edit.CharLeftExtend | Shift+Left Arrow |
| Edit.CharRight | Right Arrow |
| Edit.CharRightExtend | Shift+Right Arrow |
| Edit.DocumentEnd | End |
| Edit.DocumentEndExtend | Shift+End |
| Edit.DocumentStart | Home |
| Edit.DocumentStartExtend | Shift+Home |
| Edit.InsertTab | Tab |
| Edit.LineDown | Down Arrow |
| Edit.LineDownExtend | Shift+Up Arrow |
| Edit.LineUp | Up Arrow |
| Edit.LineUpExtend | Shift+Down Arrow |
| Edit.MoveControlDown | Ctrl+Down Arrow |
| Edit.MoveControlLeft | Ctrl+Left Arrow |
| Edit.MoveControlRight | Ctrl+Right Arrow |
| Edit.MoveControlUp | Ctrl+Up Arrow |
| Edit.SelectionCancel | Escape |
| Edit.SizeControlDown | Ctrl+Shift+Down Arrow |
| Edit.SizeControlLeft | Ctrl+Shift+Left Arrow |
| Edit.SizeControlRight | Ctrl+Shift+Right Arrow |
| Edit.SizeControlUp | Ctrl+Shift+Up Arrow |
| Edit.TabLeft | Shift+Tab |

Work Item Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------|--------------------|
| Edit.CreateCopyofWorkItem | Shift+Alt+C |
| Edit.RefreshWorkItem | F5 |
| Team.NewLinkedWorkItem | Shift+Alt+L |

Work Item Query View

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------|------------------------|
| Edit.CreateCopyofWorkItem | Shift+Alt+C |
| Edit.Indent | Shift+Alt+ Right Arrow |
| Edit.Outdent | Shift+Alt+ Left Arrow |
| Team.NewLinkedWorkItem | Shift+Alt+L |
| Team.Refresh | F5 |
| Window.Toggle | Shift+Alt+V |

Work Item Results View

| COMMANDS | KEYBOARD SHORTCUTS |
|---------------------------|------------------------|
| Edit.CreateCopyofWorkItem | Shift+Alt+C |
| Edit.Indent | Shift+Alt+ Right Arrow |
| Edit.Outdent | Shift+Alt+ Left Arrow |
| Team.GotoNextWorkItem | Shift+Alt+N |
| Team.GotoPreviousWorkItem | Shift+Alt+P |
| Team.NewLinkedWorkItem | Shift+Alt+L |
| Team.Refresh | F5 |
| Window.Toggle | Shift+Alt+V |

Workflow Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|----------|--------------------|
| | |

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------------|---|
| Edit.CompleteWord | <p>Ctrl+K, W</p> <p>or</p> <p>Ctrl+K, Ctrl+W</p> <p>or</p> <p>Ctrl+Spacebar</p> <p>or</p> <p>Alt+ Right Arrow</p> |
| Edit.DecreaseFilterLevel | Alt+, |
| Edit.IncreaseFilterLevel | Alt+. |
| Edit.ListMembers | <p>Ctrl+K, L</p> <p>or</p> <p>Ctrl+K, Ctrl+L</p> <p>or</p> <p>Ctrl+J</p> |
| Edit.ParameterInfo | <p>Ctrl+K, P</p> <p>or</p> <p>Ctrl+K, Ctrl+P</p> <p>or</p> <p>Ctrl+Shift+Spacebar</p> |
| Edit.QuickInfo | <p>Ctrl+K, I</p> <p>or</p> <p>Ctrl+K, Ctrl+I</p> |
| WorkflowDesigner.Collapse | <p>Ctrl+E, Ctrl+C</p> <p>or</p> <p>Ctrl+E, C</p> |
| WorkflowDesigner.CollapseAll | or |
| WorkflowDesigner.ConnectNodes | <p>Ctrl+E, Ctrl+F</p> <p>or</p> <p>Ctrl+E, F</p> |

| COMMANDS | KEYBOARD SHORTCUTS |
|---|-----------------------------------|
| WorkflowDesigner.CreateVariable | Ctrl+E, Ctrl+N or Ctrl+E, N |
| WorkflowDesigner.ExpandAll | Ctrl+E, Ctrl+X or Ctrl+E, X |
| WorkflowDesigner.ExpandInPlace | Ctrl+E, Ctrl+E or Ctrl+E, E |
| WorkflowDesigner.GoToParent | Ctrl+E, Ctrl+P or Ctrl+E, P |
| WorkflowDesigner.MoveFocus | Ctrl+E, Ctrl+M or Ctrl+E, M |
| WorkflowDesigner.NavigateThroughDesigner | Ctrl+Alt+F6 |
| WorkflowDesigner.Restore | Ctrl+E, Ctrl+R or Ctrl+E, R |
| WorkflowDesigner>ShowHideArgumentDesigner | Ctrl+E, Ctrl+A or Ctrl+E, A |
| WorkflowDesigner>ShowHideImportsDesigner | Ctrl+E, Ctrl+I or Ctrl+E, I |
| WorkflowDesigner>ShowHideOverviewMap | Ctrl+E, Ctrl+O or Ctrl+E, O |

| COMMANDS | KEYBOARD SHORTCUTS |
|---|-----------------------------------|
| WorkflowDesigner.ShowHideVariableDesigner | Ctrl+E, Ctrl+V or Ctrl+E, V |
| WorkflowDesigner.ToggleSelection | Ctrl+E, Ctrl+S or Ctrl+E, S |
| WorkflowDesigner.ZoomIn | Ctrl+Num + |
| WorkflowDesigner.ZoomOut | Ctrl+Num - |

XAML UI Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|------------------------------|--------------------|
| Design.FitAll | Ctrl+0 |
| Design>ShowHandles | F9 |
| Design.ZoomIn | Ctrl+Alt+= |
| Design.ZoomOut | Ctrl+Alt+- |
| Designer options | Ctrl+Shift+; |
| Format>EditText | F2 |
| Format>ResetLayout.All | Ctrl+Shift+R |
| Run project code | Ctrl+F9 |
| Timeline.Hide (Blend only) | Ctrl+H |
| Timeline.Lock (Blend only) | Ctrl+L |
| Timeline>Show (Blend only) | Ctrl+Shift+H |
| Timeline.Unlock (Blend only) | Ctrl+Shift+L |
| View>EdgeLeftMoveLeft | Ctrl+Shift+, |
| View>EdgeLeftMoveRight | Ctrl+Shift+. |
| View>EdgeRightMoveLeft | Ctrl+Shift+Alt+, |
| View>EdgeRightMoveRight | Ctrl+Shift+Alt+. |

| COMMANDS | KEYBOARD SHORTCUTS |
|-----------------------------|--------------------|
| View.ShowPropertyMarkerMenu | Ctrl+Spacebar |

XML (Text) Editor

| COMMANDS | KEYBOARD SHORTCUTS |
|-------------------------------|--------------------|
| XML.StartXSLTDebugging | Alt+F5 |
| XML.StartXSLTWithoutDebugging | Ctrl+Alt+F5 |

XML Schema Designer

| COMMANDS | KEYBOARD SHORTCUTS |
|---|--------------------|
| GraphView.BottomtoTop | Alt+Up Arrow |
| GraphView.LefttoRight | Alt+Right Arrow |
| GraphView.RighttoLeft | Alt+Left Arrow |
| GraphView.TopBottom | Alt+Down Arrow |
| OtherContextMenus.GraphView.RemovefromWorkspace | Delete |
| XsdDesigner>ShowContentModelView | Ctrl+2 |
| XsdDesigner>ShowGraphView | Ctrl+3 |
| XsdDesigner>ShowStartView | Ctrl+1 |

See also

[Image Editor for Icons](#)

[Using IntelliSense](#)

Using IntelliSense in Visual Studio

2/8/2018 • 4 min to read • [Edit Online](#)

IntelliSense is the general term for a number of features: List Members, Parameter Info, Quick Info, and Complete Word. These features help you to learn more about the code you are using, keep track of the parameters you are typing, and add calls to properties and methods with only a few keystrokes.

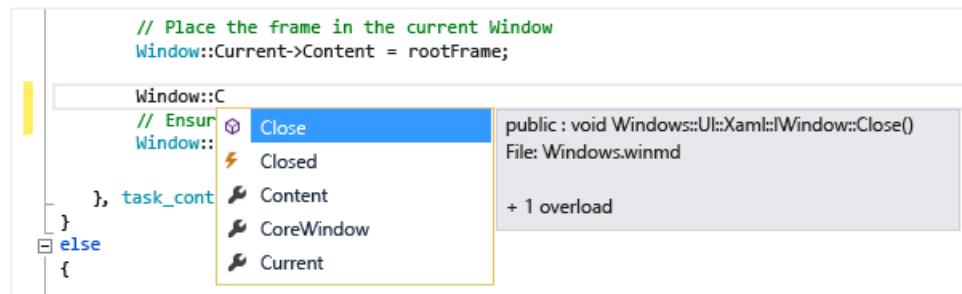
Many aspects of IntelliSense are language-specific. For more information about IntelliSense for different languages, see the topics listed in the [See also](#) section.

List Members

A list of valid members from a type (or namespace) appears after you type a trigger character (for example, a period (.) in managed code or :: in C++). If you continue typing characters, the list is filtered to include only the members that begin with those characters or where the beginning of *any* word within the name starts with those characters. IntelliSense also performs "camel case" matching, so you can just type the first letter of each camel-cased word in the member name to see the matches.

After selecting an item, you can insert it into your code by pressing **Tab** or by typing a space. If you select an item and type a period, the item appears followed by the period, which brings up another member list. When you select an item but before you insert it, you get Quick Info for the item.

In the member list, the icon to the left represents the type of the member, such as namespace, class, function, or variable. For a list of icons, see [Class View and Object Browser Icons](#). The list may be quite long, so you can press **PgUp** and **PgDn** to move up or down in the list.



You can invoke the **List Members** feature manually by typing **CTRL + J**, choosing **Edit > IntelliSense > List Members**, or by choosing the **List Members** button on the editor toolbar. When it is invoked on a blank line or outside a recognizable scope, the list displays symbols in the global namespace.

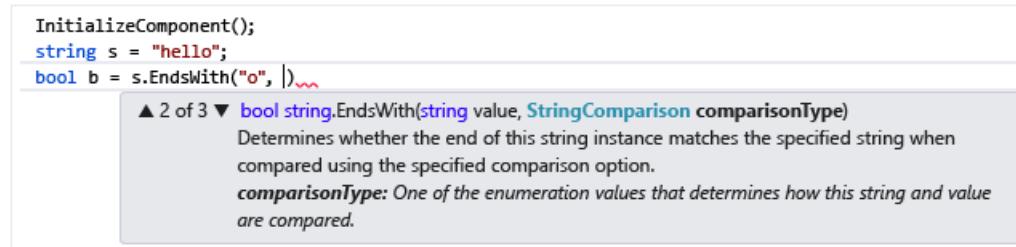
To turn List Members off by default (so that it does not appear unless specifically invoked), go to **Tools > Options > All Languages** and deselect **Auto list members**. If you want to turn off List Members only for a specific language, go to the **General** settings for that language.

You can also change to suggestion mode, in which only the text you type is inserted into the code. For example, if you enter an identifier that is not in the list and press **Tab**, in completion mode the entry would replace the typed identifier. To toggle between completion mode and suggestion mode, press **Ctrl + Alt + SPACEBAR**, or choose **Edit > IntelliSense > Toggle Completion Mode**.

Parameter Info

Parameter Info gives you information about the number, names, and types of parameters required by a method, attribute generic type parameter (in C#), or template (in C++).

The parameter in bold indicates the next parameter that is required as you type the function. For overloaded functions, you can use the UP and DOWN arrow keys to view alternative parameter information for the function overloads.

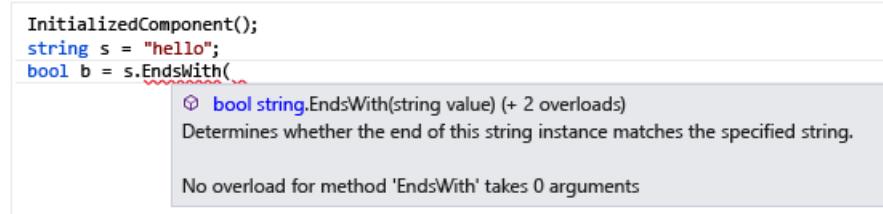


When you annotate functions and parameters with XML Documentation comments, the comments will display as Parameter Info. For more information, see [Supplying XML Code Comments](#).

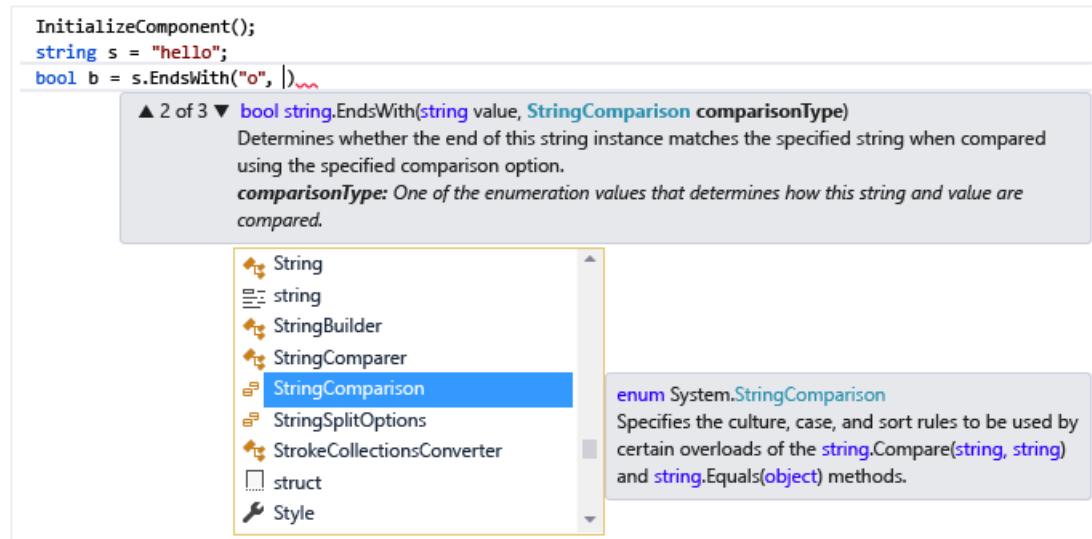
You can manually invoke Parameter Info by choosing **Edit > IntelliSense > Parameter Info**, by pressing **Ctrl + Shift + SPACE**, or by choosing the **Parameter Info** button on the editor toolbar.

Quick Info

Quick Info displays the complete declaration for any identifier in your code.



When you select a member from the **List Members** box, Quick Info also appears.



You can manually invoke Quick Info by choosing **Edit > IntelliSense > Quick Info**, by pressing **Ctrl + I**, or by choosing the **Quick Info** button on the editor toolbar.

If a function is overloaded, IntelliSense may not display information for all forms of the overload.

You can turn Quick Info off for C++ code by navigating to **Tools > Options > Text Editor > C/C++ > Advanced**, and setting **Auto Quick Info** to `false`.

Complete Word

Complete Word completes the rest of a variable, command, or function name after you have entered enough

characters to disambiguate the term. You can invoke Complete Word by choosing **Edit > IntelliSense > Complete Word**, by pressing **Ctrl+Space**, or by choosing the **Complete Word** button on the editor toolbar.

IntelliSense Options

IntelliSense options are on by default. To turn them off, choose **Tools > Options > Text Editor** and deselect **Parameter information** or **Auto list members** if you do not want the List Members feature.

Troubleshooting IntelliSense

The IntelliSense options may not work as you expect in certain cases.

The cursor is below a code error. You might not be able to use IntelliSense if an incomplete function or other error exists in the code above the cursor because IntelliSense might not be able to parse the code elements. You can resolve this problem by commenting out the applicable code.

The cursor is in a code comment. You can't use IntelliSense if the cursor is in a comment in your source file.

The cursor is in a string literal. You can't use IntelliSense if the cursor is in the quotation marks around a string literal, as in the following example:

```
MessageBox( hWnd, "String literal|")
```

The automatic options are turned off. By default, IntelliSense works automatically, but you can disable it. Even if automatic statement completion is disabled, you can invoke an IntelliSense feature.

See also

[Visual Basic IntelliSense](#)

[C# IntelliSense](#)

[JavaScript IntelliSense](#)

[Writing and refactoring code \(C++\)](#)

[Supplying XML code comments](#)

IntelliSense for Visual Basic code files

12/22/2017 • 1 min to read • [Edit Online](#)

The Visual Basic source code editor offers the following IntelliSense features:

Syntax tips

Syntax tips display the syntax of the statement that you are typing. This is useful for statements such as [Declare](#).

Automatic completion

- Completion on various keywords

For example, if you type `goto` and a space, IntelliSense displays a list of the defined labels in a drop-down menu. Other supported keywords include `Exit`, `Implements`, `Option`, and `Declare`.

- Completion on `Enum` and `Boolean`

When a statement will refer to a member of an enumeration, IntelliSense displays a list of the members of the `Enum`. When a statement will refer to a `Boolean`, IntelliSense displays a true-false drop-down menu.

Completion can be turned off by default by deselecting **Auto list members** from the **General** property page in the **Visual Basic** folder.

You can manually invoke completion by invoking List Members, Complete Word, or ALT+RIGHT ARROW. For more information, see [Using IntelliSense](#).

IntelliSense in Zone

IntelliSense in Zone assists Visual Basic developers who need to deploy applications through ClickOnce and are constrained to partial trust settings. This feature:

- Enables you to choose the permissions the application will run with.
- Display APIs in the chosen Zone as available in List Members, and display APIs that require additional permissions as unavailable.

For more information, see [Code Access Security for ClickOnce Applications](#).

Filtered completion lists

In Visual Basic, IntelliSense completion lists have two tab controls located near the bottom of the lists. The **Common** tab, which is selected by default, displays items that are most often used to complete the statement that you are writing. The **All** tab displays all items that are available for automatic completion, including those that are also on the **Common** tab.

See also

[Using IntelliSense](#)

4 min to read •

C# IntelliSense

1/26/2018 • 7 min to read • [Edit Online](#)

C# IntelliSense is available when coding in the editor, and while debugging in the [Immediate Mode](#) command window.

Completion lists

The IntelliSense completion lists in C# contain tokens from List Members, Complete Word, and more. It provides quick access to:

- Members of a type or namespace
- Variables, commands, and functions names
- Code snippets
- Language Keywords
- Extension Methods

The Completion List in C# is also smart enough to filter out irrelevant tokens and pre-select a token based on context. For more information, see [Filtered Completion Lists](#).

Code Snippets in Completion Lists

In C#, the completion list includes code snippets to help you easily insert predefined bodies of code into your program. Code snippets appear in the completion list as the snippet's [shortcut text](#). For more information about code snippets that are available in C# by default, see [C# Code Snippets](#).

Language Keywords in Completion Lists

In C#, the completion list also includes language keywords. For more information about C# language keywords, see [C# Keywords](#).

Extension Methods in Completion Lists

In C#, the completion list includes Extension Methods that are in scope.

NOTE

The completion list does not display all extension methods for [String](#) objects.

Extension methods use a different icon than instance methods. For a listing of list icons, see [Class View and Object Browser Icons](#). When an instance method and extension method with the same name are both in scope, the completion list displays the extension method icon.

Filtered Completion Lists

IntelliSense removes unnecessary members from the completion list by using filters. C# filters the completion lists that appear for these items:

- **Interfaces and base classes:** IntelliSense automatically removes items from the interface and base class completion lists, in both class declaration base and interface lists and constraint lists. For example, enums do not appear in the completion list for base classes, because enums cannot be used for base classes. The completion list of base classes only contains interfaces and namespaces. If you select an item in the list and then type a comma, IntelliSense removes base classes from the completion list because C# does not support multiple inheritance. The same behavior occurs for constraint clauses also.
- **Attributes:** When you apply an attribute to a type, the completion list is filtered so that the list only contains those types that descend from the namespaces that contain those types, such as [Attribute](#).
- **Catch clauses**
- **Object initializers:** Only members that can be initialized will appear in the completion list.
- **new keyword:** When you type `new` and then press the SPACEBAR, a completion list appears. An item is automatically selected in the list, based on the context in your code. For example, items are automatically selected in the completion list for declarations and for return statements in methods.
- **enum keyword:** When you press the SPACEBAR after an equal sign for an enum assignment, a completion list appears. An item is automatically selected in the list, based on the context in your code. For example, items are automatically selected in the completion list after you type the keyword `return` and when you make a declaration.
- **as and is operators:** A filtered completion list is displayed automatically when you press the SPACEBAR after you have typed the `as` or `is` keyword.
- **Events:** When you type the keyword `event`, the completion list only contains delegate types.
- **Parameter help** automatically sorts to the first method overload that matches the parameters as you enter them. If multiple method overloads are available, you can use the up and down arrows to navigate to the next possible overload in the list.

Most recently used members

IntelliSense remembers the members that you have recently selected in the pop-up [List Members](#) box for automatic object name completion. The next time you use Member List, the most recently used members are shown at the top. The history of most recently used members is cleared between each session in the IDE.

override

When you type `override` and then press SPACEBAR, IntelliSense displays all of the valid base class members that you can override in a pop-up list box. Typing the return type of the method after `override` will prompt IntelliSense to only show methods that return the same type. When IntelliSense cannot find any matches, it will display all of the base class members.

Automatic Code Generation

Add using

The **Add using** IntelliSense operation automatically adds the required `using` directive to your code file. This feature enables you to maintain your focus on the code you are writing rather than requiring you to shift your focus to another part of the code.

To initiate the Add using operation, position the cursor on a type reference that cannot be resolved. For example, when you create a console application and then add `Xm1TextReader` to the body of the `Main` method, a red squiggle appears on that line of code because the type reference cannot be resolved. You can then invoke the Add using through the Quick Action. The Quick Action is only visible when the cursor is positioned on the unbound

type.



Click the light bulb icon, and then choose **using System.Xml;** to automatically add the using directive.

Remove and sort usings

The **Remove and Sort Usings** option sorts and removes `using` and `extern` declarations without changing the behavior of the source code. Over time, source files may become bloated and difficult to read because of unnecessary and unorganized `using` directives. The **Remove and Sort Usings** option compacts source code by removing unused `using` directives, and improves readability by sorting them. On the **Edit** menu, choose **IntelliSense**, and then choose **Organize Usings**.

Implement Interface

IntelliSense provides an option to help you implement an [interface](#) while working in the Code Editor. Normally, to implement an interface properly, you must create a method declaration for every member of the interface in your class. Using IntelliSense, after you type the name of an interface in a class declaration, a Quick Actions light bulb is displayed. The light bulb gives you the option to implement the interface automatically, using explicit or implicit naming. Under explicit naming, the method declarations carry the name of the interface; under implicit naming, the method declarations do not indicate the interface to which they belong. An explicitly named interface method can only be accessed through an interface instance, and not through a class instance. For more information, see [Explicit Interface Implementation](#).

Implement Interface will generate the minimum number of method stubs that is required to satisfy the interface. If a base class implements parts of the interface, then those stubs are not regenerated.

Implement abstract base class

IntelliSense provides an option to help you implement members of an abstract base class automatically while working in the Code Editor. Normally, to implement members of an abstract base class requires creating a new method definition for each method of the abstract base class in your derived class. Using IntelliSense, after typing the name of an abstract base class in a class declaration, a Quick Actions light bulb is displayed. The light bulb gives you the option to implement the base class methods automatically.

The method stubs that are generated by the Implement Abstract Base Class feature are modeled by the code snippet defined in the file `MethodStub.snippet`. Code Snippets are modifiable. For more information, see [Walkthrough: Creating a Code Snippet](#).

Generate from usage

The **Generate From Usage** feature enables you to use classes and members before you define them. You can generate a stub for any class, constructor, method, property, field, or enum that you want to use but have not yet defined. You can generate new types and members without leaving your current location in code. This minimizes interruption to your workflow.

A red wavy underline appears under each undefined identifier. When you rest the mouse pointer on the identifier, an error message appears in a tooltip. To display the appropriate options, you can use one of the following procedures:

- Click the undefined identifier. A Quick Actions light bulb appears under the identifier. Click the light bulb.
- Click the undefined identifier, and then press **Ctrl + .** (Ctrl + period).
- Right-click the undefined identifier, and then click **Quick Actions and Refactorings**.

The options that appear can include the following:

- **Generate property**
- **Generate field**
- **Generate method**
- **Generate class**
- **Generate new type...** (for a class, struct, interface, or enum)

Generate event handlers

In the Code Editor, IntelliSense can help you hook up methods (event handlers) to event fields.

When you type the `+ =` operator after an event field in a .cs file, IntelliSense prompts you with the option to press the **Tab** key. This inserts a new instance of a delegate that points to the method handling the event.

```
button1.Click +=  
    new EventHandler(button1_Click); (Press TAB to insert)
```

If you press **Tab**, IntelliSense automatically finishes the statement for you and displays the event handler reference as selected text in the Code Editor. To complete the automatic event hookup, IntelliSense prompts you to press the **Tab** key again to create an empty stub for the event handler.

```
button1.Click +=new EventHandler(button1_Click);  
    Press TAB to generate handler 'button1_Click' in this class
```

NOTE

If a new delegate that is created by IntelliSense references an existing event handler, IntelliSense communicates this information in the tooltip. You can then modify this reference; the text is already selected in the Code Editor. Otherwise, automatic event hookup is complete at this point.

If you press **Tab**, IntelliSense stubs out a method with the correct signature and puts the cursor in the body of your event handler.

NOTE

Use the **Navigate Backward** command on the **View** menu (**Ctrl + -**) to go back to the event hookup statement.

See also

[Using IntelliSense](#)
[Visual Studio IDE](#)

JavaScript IntelliSense

1/8/2018 • 4 min to read • [Edit Online](#)

Visual Studio 2017 provides a powerful JavaScript editing experience right out of the box. Powered by a TypeScript based language service, Visual Studio delivers richer IntelliSense, support for modern JavaScript features, and improved productivity features such as Go to Definition, refactoring, and more.

NOTE

The JavaScript language service in Visual Studio 2017 uses a new engine for the language service (called "Salsa"). Details are included in this topic, and you can also read this [blog post](#). The new editing experience also mostly applies to Visual Studio Code. See the [VS Code docs](#) for more info.

For more information about the general IntelliSense functionality of Visual Studio, see [Using IntelliSense](#).

What's New in the JavaScript language service in Visual Studio 2017

Starting in Visual Studio 2017, JavaScript IntelliSense displays a lot more information on parameter and member lists. This new information is provided by the TypeScript language service, which uses static analysis behind the scenes to better understand your code. TypeScript uses several sources to build up this information:

- [IntelliSense based on type inference](#)
- [IntelliSense based on JSDoc](#)
- [IntelliSense based on TypeScript declaration files](#)
- [Automatic acquisition of type definitions](#)

IntelliSense based on type inference

In JavaScript, most of the time there is no explicit type information available. Luckily, it is usually fairly easy to figure out a type given the surrounding code context. This process is called type inference.

For a variable or property, the type is typically the type of the value used to initialize it or the most recent value assignment.

```
var nextItem = 10;
nextItem; // here we know nextItem is a number

nextItem = "box";
nextItem; // now we know nextItem is a string
```

For a function, the return type can be inferred from the return statements.

For function parameters, there is currently no inference, but there are ways to work around this using JSDoc or TypeScript `.d.ts` files (see later sections).

Additionally, there is special inference for the following:

- "ES3-style" classes, specified using a constructor function and assignments to the prototype property.
- CommonJS-style module patterns, specified as property assignments on the `exports` object, or assignments to the `module.exports` property.

```
function Foo(param1) {
  this.prop = param1;
}
Foo.prototype.getIt = function () { return this.prop; };
// Foo will appear as a class, and instances will have a 'prop' property and a 'getIt' method.

exports.Foo = Foo;
// This file will appear as an external module with a 'Foo' export.
// Note that assigning a value to "module.exports" is also supported.
```

IntelliSense based on JSDoc

Where type inference does not provide the desired type information (or to support documentation), type information may be provided explicitly via JSDoc annotations. For example, to give a partially declared object a specific type, you can use the `@type` tag as shown below:

```
/** 
 * @type {{a: boolean, b: boolean, c: number}}
 */
var x = {a: true};
x.b = false;
x. // <- "x" is shown as having properties a, b, and c of the types specified
```

As mentioned, function parameters are never inferred. However, using the JSDoc `@param` tag you can add types to function parameters as well.

```
/** 
 * @param {string} param1 - The first argument to this function
 */
function Foo(param1) {
  this.prop = param1; // "param1" (and thus "this.prop") are now of type "string".
}
```

See [this doc](#) for the JsDoc annotations currently supported.

IntelliSense based on TypeScript Declaration Files

Because JavaScript and TypeScript are now based on the same language service, they are able to interact in a richer way. For example, JavaScript IntelliSense can be provided for values declared in a `.d.ts` file ([more info](#)), and types such as interfaces and classes declared in TypeScript are available for use as types in JsDoc comments.

Below, we show a simple example of a TypeScript definition file providing such type information (via an interface) to a JavaScript file in the same project (using a JsDoc tag).

```
1  /**
2   * @param {Person} emp - The person to set
3   */
4   */
5   function setEmployee(emp) {
6     emp.
7     address
8     age
9     emp
10    setEmployee
11  }
12 }
```

```
5   declare interface Person {
6     age: number;
7     address: {
8       street: string;
9       zip: number;
10    }
11 }
```

Automatic acquisition of type definitions

In the TypeScript world, most popular JavaScript libraries have their APIs described by `.d.ts` files, and the most common repository for such definitions is on [DefinitelyTyped](#).

By default, the Salsa language service will try to detect which JavaScript libraries are in use and automatically download and reference the corresponding `.d.ts` file that describes the library in order to provide richer IntelliSense. The files are downloaded to a cache located under the user folder at

```
%LOCALAPPDATA%\Microsoft\TypeScript .
```

NOTE

This feature is **disabled** by default if using a `tsconfig.json` configuration file, but may be set to enabled as outlined further below.

Currently auto-detection works for dependencies downloaded from npm (by reading the `package.json` file), Bower (by reading the `bower.json` file), and for loose files in your project that match a list of roughly the top 400 most popular JavaScript libraries. For example, if you have `jquery-1.10.min.js` in your project, the file `jquery.d.ts` will be fetched and loaded in order to provide a better editing experience. This `.d.ts` file will have no impact on your project.

If you do not wish to use auto-acquisition, disable it by adding a configuration file as outlined below. You can still place definition files for use directly within your project manually.

See also

[Using IntelliSense](#)

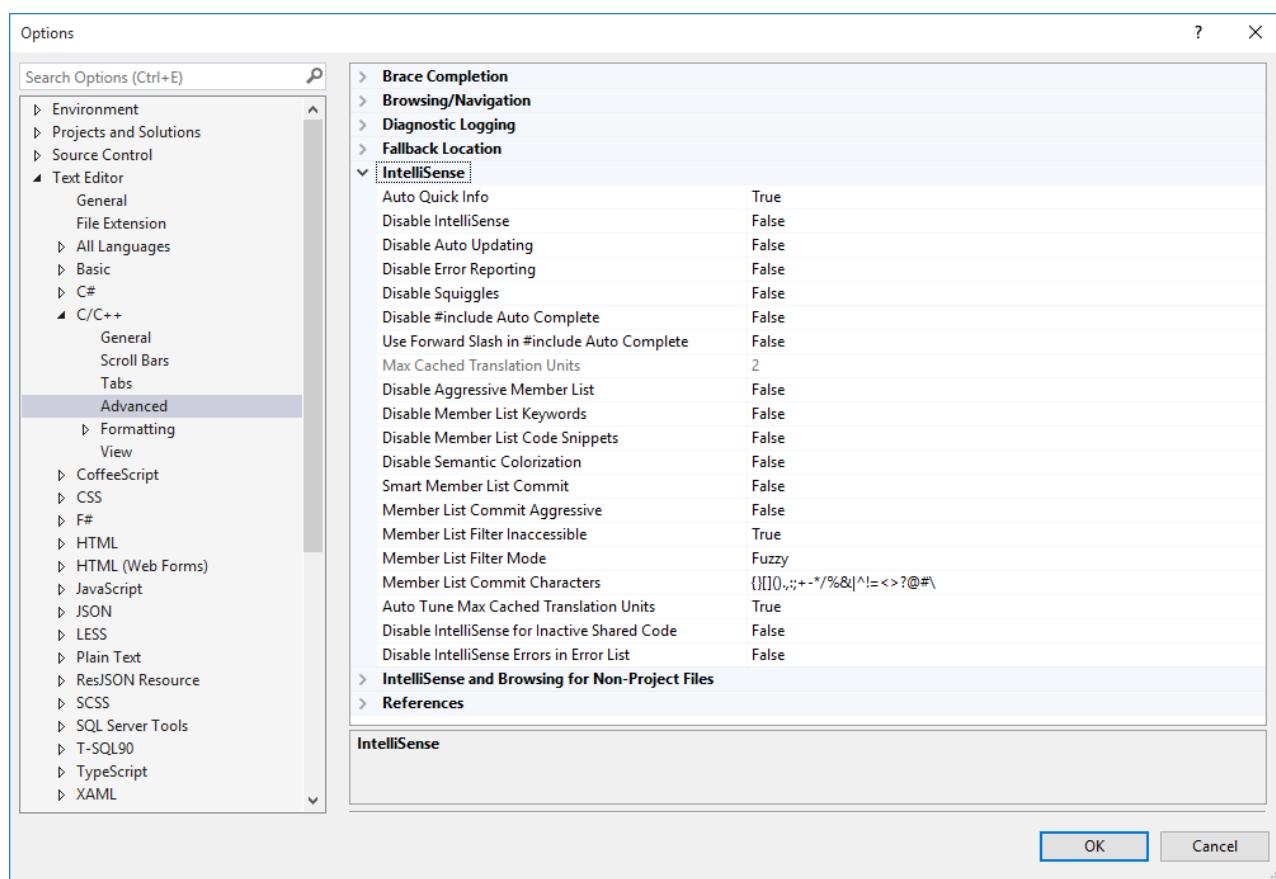
Visual C++ IntelliSense

3/1/2018 • 4 min to read • [Edit Online](#)

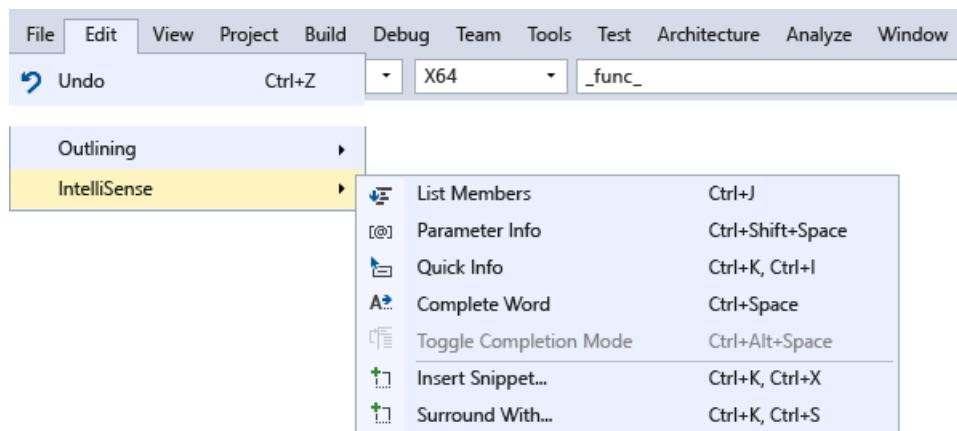
IntelliSense for C++ is available for stand-alone files as well as for files that are part of a C++ project. In cross-platform projects, some IntelliSense features are available in .cpp and .c files in the shared code project, even when you are in an Android or iOS context.

IntelliSense features in C++

IntelliSense is a name given to a set of features that make coding more convenient. Since different people have different ideas about what is convenient, virtually all of the IntelliSense features can be enabled or disabled in the **Options** dialog box, under **Text Editor > C/C++ > Advanced**. The **Options** dialog box is available from the **Tools** menu on the menu bar.



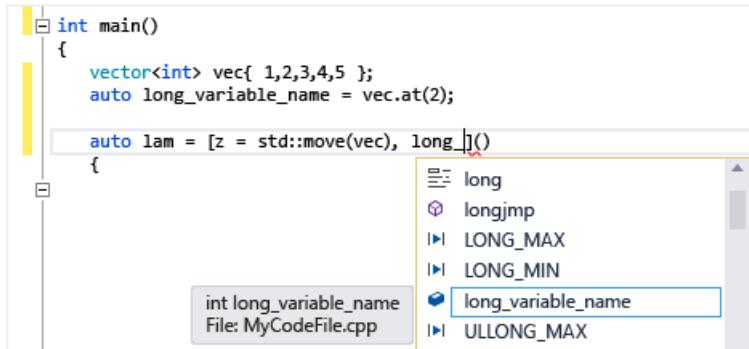
You can use the menu items and keyboard shortcuts shown in the following image to access IntelliSense.



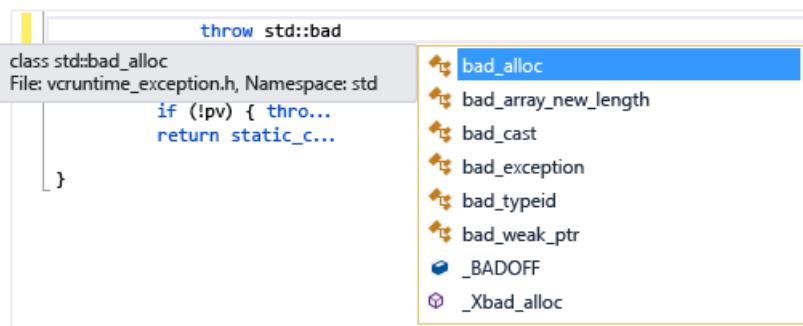
Statement completion and member list

When you start typing a keyword, type, function, variable name, or other program element that the compiler recognizes, the editor offers to complete the word for you.

For a list of the icons and their meanings, see [Class View and Object Browser Icons](#).

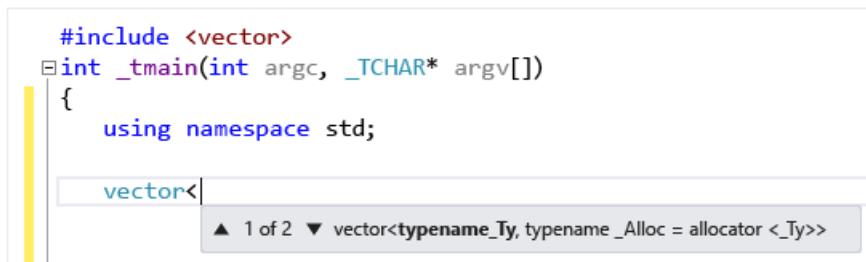


The first time member list is invoked it only shows members that are accessible for the current context. If you press **Ctrl+J** after that, it shows all members regardless of accessibility. If you invoke it a third time, an even wider list of program elements is shown. You can turn off member list in the **Options** dialog box, under **Text Editor > C/C++ > General > Auto list members**.



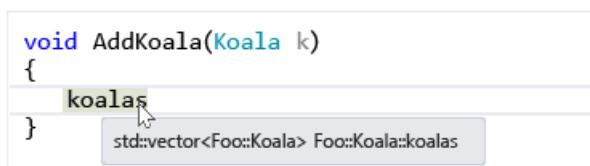
Parameter help

When you type an opening brace of a function call, or angle bracket on a class template variable declaration, the editor shows a small window with the parameter types for each overload of the function or constructor. The "current" parameter—based on the cursor location—is in bold. You can turn off parameter information in the **Options** dialog box, under **Text Editor > C/C++ > General > Parameter information**.



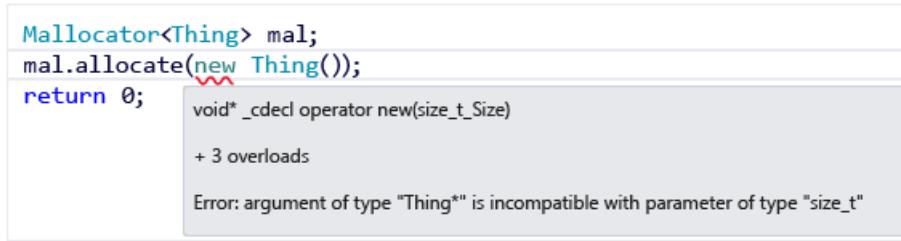
Quick Info

When you hover the mouse cursor over a variable, a small window appears inline that shows the type information and the header in which the type is defined. Hover over a function call to see the function's signature. You can turn off Quick Info in the **Options** dialog box, under **Text Editor > C/C++ > Advanced > Auto Quick Info**.



Error squiggles

Squiggles under a program element (variable, keyword, brace, type name, and so on) call your attention to an error or potential error in the code. A green squiggle appears when you write a forward declaration, to remind you that you still need to write the implementation. A purple squiggle appears in a shared project when there is an error in code that is not currently active, for example when you are working in the Windows context but enter something that would be an error in an Android context. A red squiggle indicates a compiler error or warning in active code that you need to deal with.



```
Mallocator<Thing> mal;
mal.allocate(new Thing());
return 0;
```

void* __cdecl operator new(size_t Size)
+ 3 overloads

Error: argument of type "Thing*" is incompatible with parameter of type "size_t"

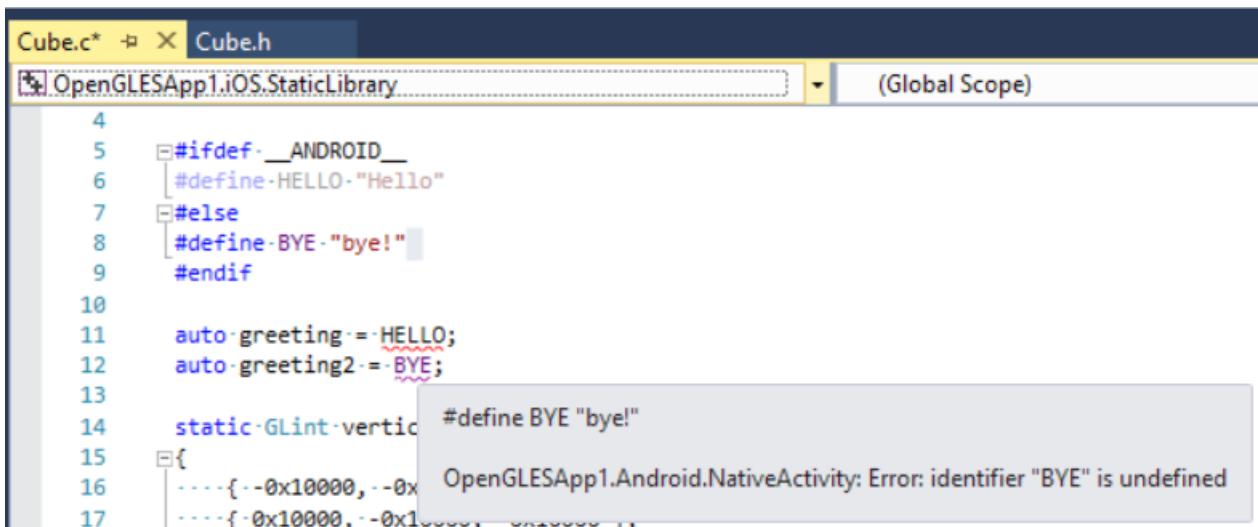
Code colorization and fonts

The default colors and fonts can be changed in the **Options** dialog box, under **Environment > Fonts and Colors**. You can change the fonts for many UI windows here, not just the editor. The settings that are specific to C++ begin with "C++"; the other settings are for all languages.

Cross-platform IntelliSense

In a shared code project, some IntelliSense features such as squiggles are available even when you are working in an Android context. If you write some code that would result in an error in an inactive project, IntelliSense still shows squiggles, but they are in a different color than squiggles for errors in the current context.

Here's an OpenGL ES Application that is configured to build for Android and iOS. The illustration shows shared code being edited. In the first image, Android is the active project:



Cube.c* → X Cube.h

OpenGLApp1.iOS.StaticLibrary (Global Scope)

```
4
5 #ifdef __ANDROID__
6     #define HELLO "Hello"
7 #else
8     #define BYE "bye!"
9 #endif
10
11 auto greeting = HELLO;
12 auto greeting2 = BYE;
13
14 static GLint vertice #define BYE "bye!"
```

OpenGLESApp1.Android.NativeActivity: Error: identifier "BYE" is undefined

Notice the following:

- The `#else` branch on line 8 is grayed out to indicate inactive region, because `_ANDROID_` is defined for Android project.
- The greeting variable at line 11 is initialized with identifier HELLO, which has a purple squiggle. This is because no identifier HELLO is defined in the currently inactive iOS project. While in Android project line 11 would compile, it won't in iOS. Since this is shared code, that is something you should change even though it compiles in the currently active configuration.
- Line 12 has red squiggle on identifier BYE; this identifier is not defined in the currently selected active project.

Now, change the active project to iOS.StaticLibrary and notice how the squiggles change.

```

4
5  #ifdef __ANDROID__
6  #define HELLO "Hello"
7  #else
8  #define BYE "bye!"
9  #endif
10
11  auto greeting = HELLO;
12  auto greeting2 = BYE;
13
14  static GLint vertexCount = 6;
15
16  #define BYE "bye!"
17

```

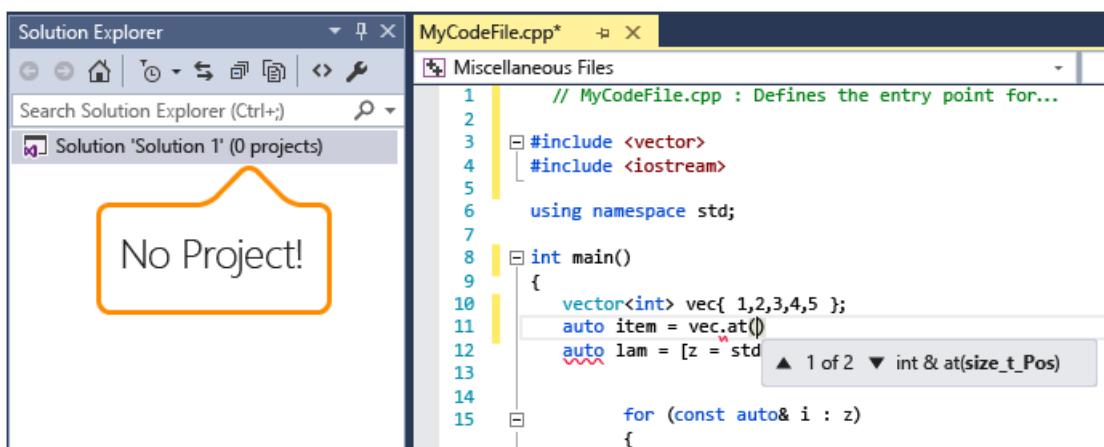
OpenGLESApp1.Android.NativeActivity: Error: identifier "BYE" is undefined

Notice the following:

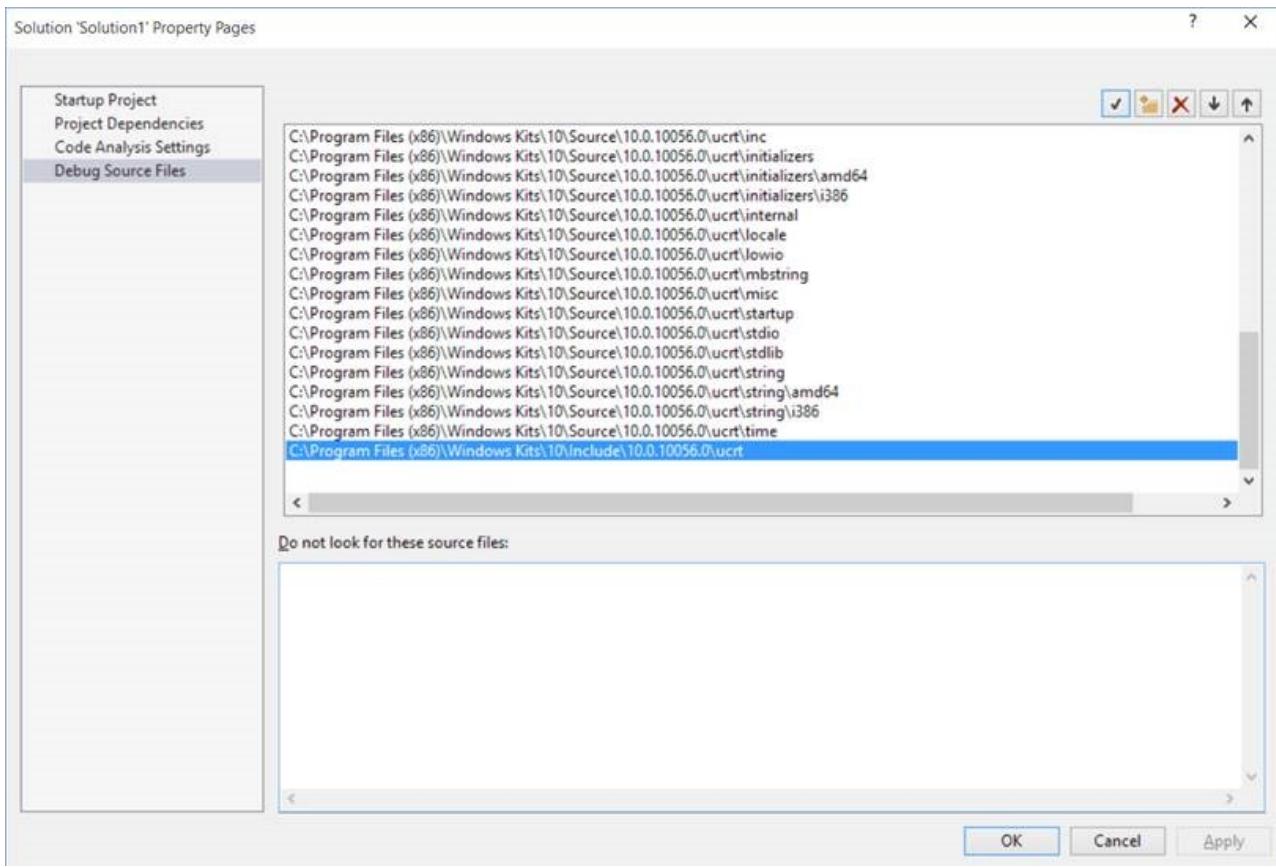
- The #ifdef branch on line 6 is grayed out to indicate inactive region, because `_ANDROID_` is not defined for iOS project.
- The greeting variable at line 11 is initialized with identifier HELLO, which now has red squiggle. This is because no identifier HELLO is defined in the currently active iOS project.
- Line 12 has purple squiggle on identifier BYE; this identifier is not defined in currently inactive `Android.NativeActivity` project.

IntelliSense for stand-alone files

When you open a single file outside of any project, you still get IntelliSense. You can enable or disable particular IntelliSense features in the **Options** dialog box, under **Text Editor > C/C++ > Advanced**. To configure IntelliSense for single files that aren't part of a project, look for the **IntelliSense and Browsing for Non-Project Files** section.



By default, single file IntelliSense only uses standard include directories to find header files. To add additional directories, open the shortcut menu on the Solution node, and add your directory to **Debug Source Code** list, as the following illustration shows:



See also

[Using IntelliSense](#)

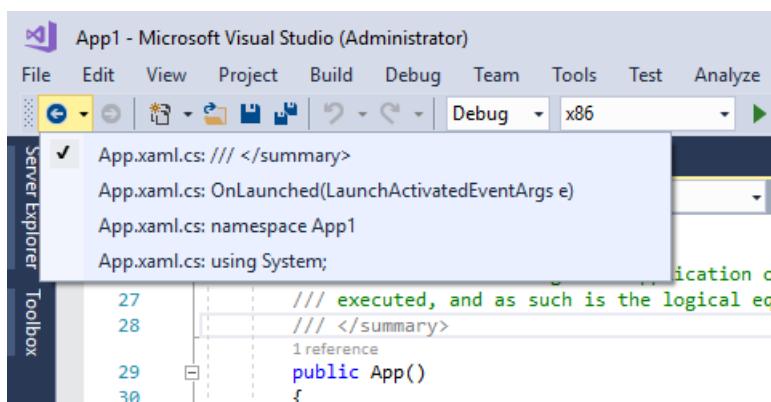
Navigate code

1/13/2018 • 5 min to read • [Edit Online](#)

Visual Studio provides numerous ways to navigate code in the editor. This topic summarizes the different ways you can navigate your code, and provides links to topics that go into more detail.

Navigate Backward and Navigate Forward commands

You can use the **Navigate Backward** (**Ctrl + -**) and **Navigate Forward** (**Ctrl + Shift + -**) buttons on the toolbar to move the insertion point to previous locations, or to return to a more recent location from a previous location. These buttons retain the last 20 locations of the insertion point. These commands are also available on the **View** menu, under **Navigate Backward** and **Navigate Forward**.



Navigation bar

You can use the **navigation bar** (the drop-down boxes at the top of the code window) to navigate to code in a codebase. You can choose a type or member to go directly to it. The navigation bar appears when you edit code in a Visual Basic, C#, or C++ code base. In a partial class, members defined outside the current code file may be disabled (they appear in gray).

```

1  using BikeSha _eventsService
2  using BikeSha _overallProgress
3  using BikeSha _remainingTime
4  using BikeSha _ridesService
5  using System; _showThanks
6  using System.
7  using System.
8  using System.
9  using System.
10 using System.
11 using Xamarin
12
13 namespace Bik _eventsService
14 {
15     public class BookingViewModel : ViewModelBase
16     {
17         private readonly IEventsService eventsService;
18         private readonly IRidesService ridesService;
19         private readonly IBookingService bookingService;
20         private readonly IEventService eventService;
21
22         public bool ShowThanks
23         {
24             get
25             {

```

You can navigate around the drop-down boxes as follows:

- To navigate to another project that the current file belongs to, choose it in the left drop-down.
- To navigate to a class or type, choose it in the middle drop-down.
- To navigate directly to a procedure or other member of a class, choose it in the right drop-down.
- To shift focus from the code window to the navigation bar, press the shortcut key combination **Ctrl + F2**.
- To shift focus from box to box on the navigation bar, press the **Tab** key.
- To select the navigation bar item that has focus and return to the code window, press the **Enter** key.
- To return focus from the navigation bar to the code without selecting anything, press the **Esc** key.

To hide the navigation bar, change the **Navigation bar** option in the Text Editor All Languages settings (**Tools, Options, Text Editor, All Languages**), or you can change the settings for individual languages.

Find All References

Finds all the references to the selected element in the solution. You can use this to check possible side-effects of a large refactoring, or to verify "dead" code. Press **F8** to jump between results. For more information, see [Finding references in your code](#).

| INPUT | FUNCTION |
|-----------------|---|
| Keyboard | Place your text cursor somewhere inside the type name, and press Shift + F12 |
| Mouse | Select Find All References from the context menu |

Reference highlighting

When you click a symbol in the source code, all instances of that symbol are highlighted in the document. The highlighted symbols may include declarations and references, and many other symbols that **Find All References**

would return. These include the names of classes, objects, variables, methods, and properties. In Visual Basic code, keywords for many control structures are also highlighted. To move to the next or the previous highlighted symbol, press **Ctrl + Shift + DOWN ARROW** or **Ctrl + Shift + UP ARROW**. You can change the highlighting color in **Tools, Options, Environment, Fonts and Colors, Highlighted Reference**.

Go To commands

Go To has the following commands, which are available in the **Edit** menu under **Go To**:

- **Go To Line (Ctrl + G)**: Move to the specified line number in the active document.
- **Go to All (Ctrl + T or Ctrl + ,)**: Move to the specified line, type, file, member, or symbol.
- **Go to File (Ctrl + 1, Ctrl + F)**: Move to the specified file in the solution.
- **Go to Type (Ctrl + 1, Ctrl + T)**: Move to the specified type in the solution.
- **Go to Member (Ctrl + 1, Ctrl + M)**: Move to the specified member in the solution.
- **Go to Symbol (Ctrl + 1, Ctrl + S)**: Move to the specified symbol in the solution.

See more about these commands in the [Find code using Go To commands](#) topic.

Go To Definition

Go To Definition takes you to the definition of the selected element. For more information, see [Go To Definition](#) and [Peek Definition](#).

| INPUT | FUNCTION |
|-----------------|---|
| Keyboard | Place your text cursor somewhere inside the type name, and press F12 |
| Mouse | Right-click on the type name and select Go To Definition OR press Ctrl and click on the type name (new for Visual Studio 2017 version 15.4) |

Peek Definition

Peek Definition displays the definition of the selected element in a window without navigating away from your current location in the code editor. For more information, see [How to: View and Edit Code by Using Peek Definition](#) and [Go To Definition and Peek Definition](#).

| INPUT | FUNCTION |
|-----------------|---|
| Keyboard | Place your text cursor somewhere inside the type name, and press Alt + F12 |
| Mouse | Right-click on the type name and select Peek Definition OR press Ctrl and click on the type name (if you have the Open definition in peek view option checked) |

Go To Implementation

Using Go To Implementation, you can navigate from a base class or type to its implementations. If there are multiple implementations, you will see them listed in the **Find Symbol Results** window:

| INPUT | FUNCTION |
|-----------------|--|
| Keyboard | Place your text cursor somewhere inside the type name, and press Ctrl + F12 |
| Mouse | Right-click on the type name and select Go To Implementation |

Call Hierarchy

You can view calls to and from a method in the [Call Hierarchy window](#):

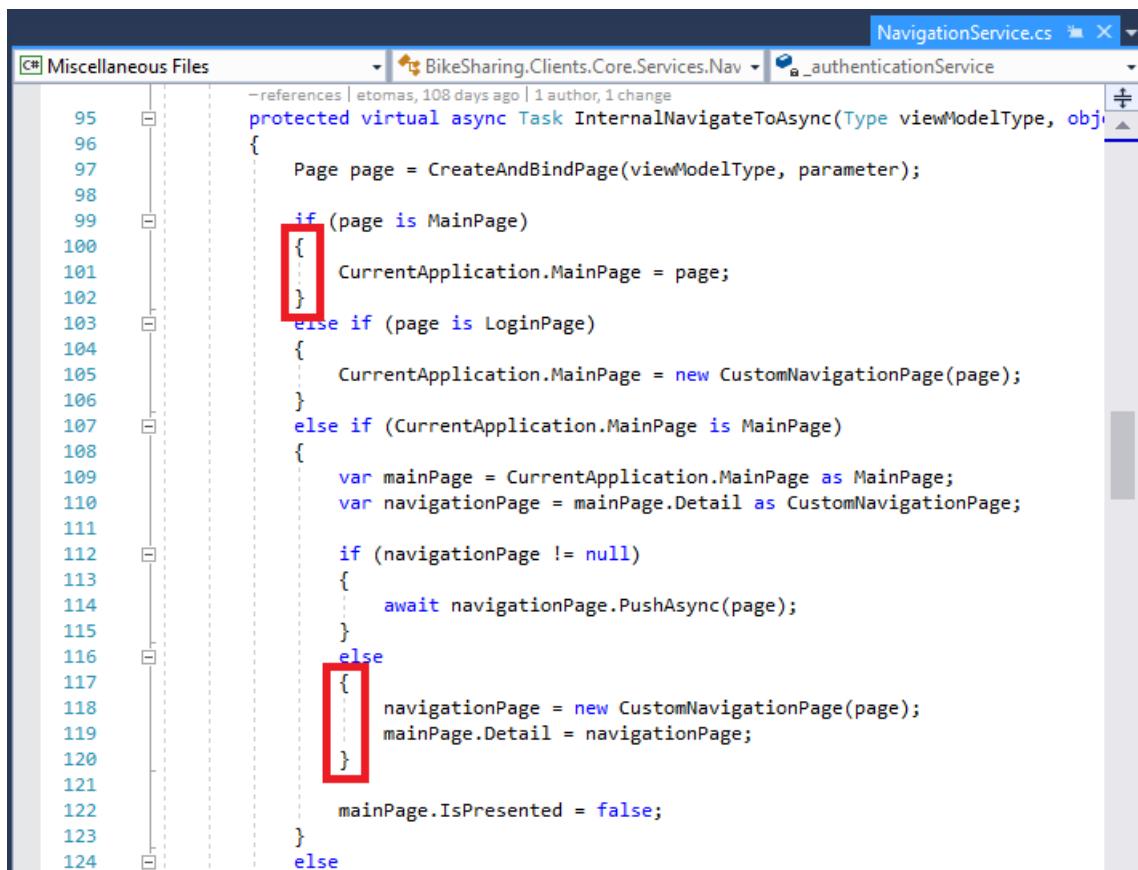
| INPUT | FUNCTION |
|-----------------|--|
| Keyboard | Place your text cursor somewhere inside the type name, and press Ctrl + K, Ctrl + T |
| Mouse | Right-click on the member name and select View Call Hierarchy |

Next Method and Previous Method commands (Visual Basic)

In Visual Basic code files, use these commands to move the insertion point to different methods. Choose **Edit, Next Method** or **Edit, Previous Method**.

Structure Visualizer

The Structure Visualizer feature in the code editor shows *structure guide lines* - vertical dashed lines that indicate matching curly braces in your codebase. This makes it easier to see where logical blocks begin and end.



```

    protected virtual async Task InternalNavigateToAsync(Type viewModelType, object parameter)
    {
        Page page = CreateAndBindPage(viewModelType, parameter);

        if (page is MainPage)
        {
            CurrentApplication.MainPage = page;
        }
        else if (page is LoginPage)
        {
            CurrentApplication.MainPage = new CustomNavigationPage(page);
        }
        else if (CurrentApplication.MainPage is MainPage)
        {
            var mainPage = CurrentApplication.MainPage as MainPage;
            var navigationPage = mainPage.Detail as CustomNavigationPage;

            if (navigationPage != null)
            {
                await navigationPage.PushAsync(page);
            }
            else
            {
                navigationPage = new CustomNavigationPage(page);
                mainPage.Detail = navigationPage;
            }

            mainPage.IsPresented = false;
        }
        else
    }
}

```

To disable structure guide lines, go to **Tools**, **Options**, **Text Editor**, **General** and clear the **Show structure guide lines** box.

Enhanced scroll bar

You can use the enhanced scroll bar in a code window to get a bird's-eye view of your code. In map mode, you can see previews of the code when you move the cursor up and down the scroll bar. For more information, see [How to: Track Your Code by Customizing the Scroll Bar](#).

CodeLens information

You can find info about specific code, like changes and who made those changes, references, bugs, work items, code reviews, and unit test status when you use CodeLens in the code editor. CodeLens works like a heads-up display when you use Visual Studio Enterprise with Team Foundation Server. See [Find code changes and other history](#).

See also

[Writing code in the code and text editor](#)

[Viewing call hierarchy](#)

Finding references in your code

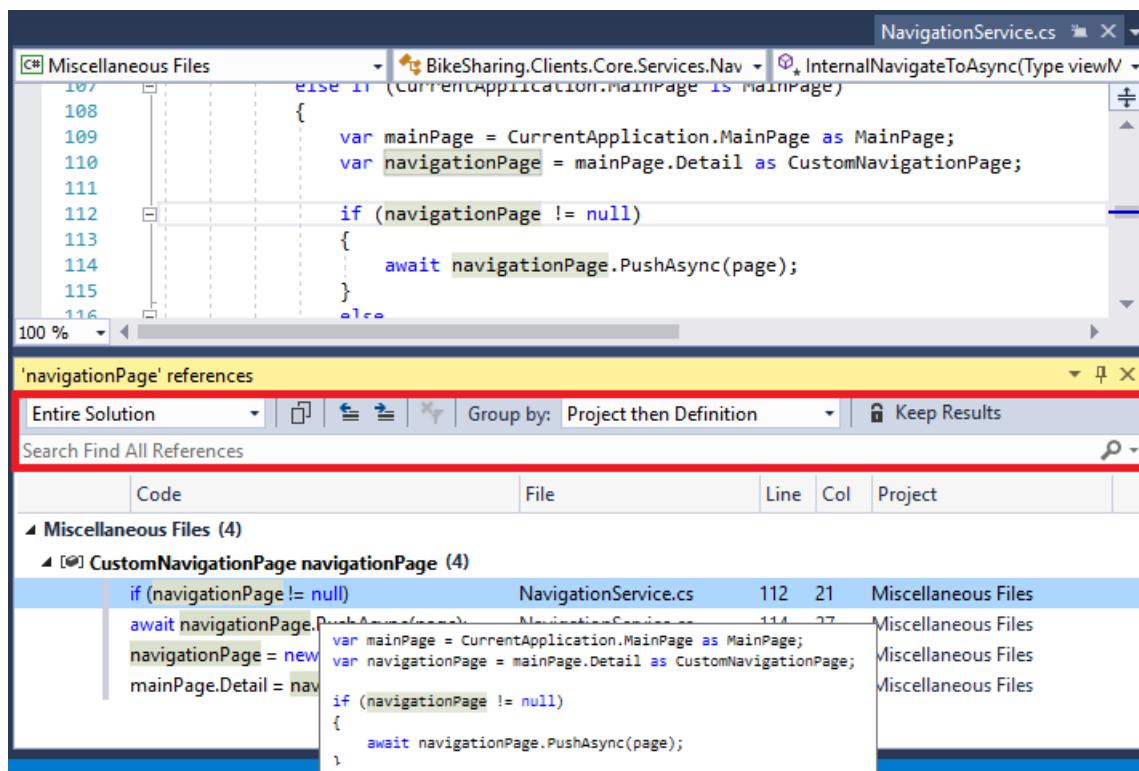
12/22/2017 • 2 min to read • [Edit Online](#)

You can use the **Find All References** command to find where particular code elements are referenced throughout your codebase. The **Find All References** command is available on the context (right-click) menu of the element you want to find references to. Or, if you are a keyboard user, press **Shift + F12**.

The results appear in a tool window named '**element** references', where *element* is the name of the item you are searching for. A toolbar in the **references** window enables you to:

- Change the scope of the search in a drop-down list box. You can choose to look only in changed documents all the way up to the entire solution.
- Copy the selected referenced item by choosing the **Copy** button.
- Choose buttons to go to the next or previous location in the list, or press the **F8** and **Shift + F8** keys to do so.
- Remove any filters on the returned results by choosing the **Clear All Filters** button.
- Change how returned items are grouped by choosing a setting in the **Group by:** drop-down list box.
- Keep the current search results window by choosing the **Keep Results** button. When you choose this button, the current search results stay in this window, and new search results appear in a new tool window.
- Search for strings within the search results by entering text in the **Search Find All References** text box.

You can also hover the mouse over any search result to see a preview of the reference.



Navigate to references

You can use the following methods to navigate to references in the **references** window:

- Press **F8** to go to the next reference, or **Shift + F8** to go to the previous reference.
- Press the **Enter** key on a reference, or double-click it, to go to it in code.
- On the context menu of a reference, choose the **Go To Previous Location** or **Go To Next Location** commands.

- Choose the **UP arrow** and **DOWN arrow** keys (if they are enabled in the Options dialog box). To enable this functionality, on the menu bar, choose **Tools**, **Options**, **Environment**, **Tabs and Windows**, **Preview Tab**, and then select the **Allow new files to be opened in the preview tab** and **Preview selected files in Find Results** boxes.

Change reference groupings

By default, references are grouped by project, then by definition. However, you can change this grouping order by changing the setting in the **Group by:** drop-down list box on the toolbar. For example, you can change it from the default setting of **Project then definition** to **Definition then project**, as well to other settings.

Definition and **Project** are the two default groupings used, but you can add others by choosing the **Grouping** command on the selected item's context menu. Adding more groupings can be helpful if your solution has a lot of files and paths.

See also

[Navigating Code](#)

View type and member definitions

3/5/2018 • 3 min to read • [Edit Online](#)

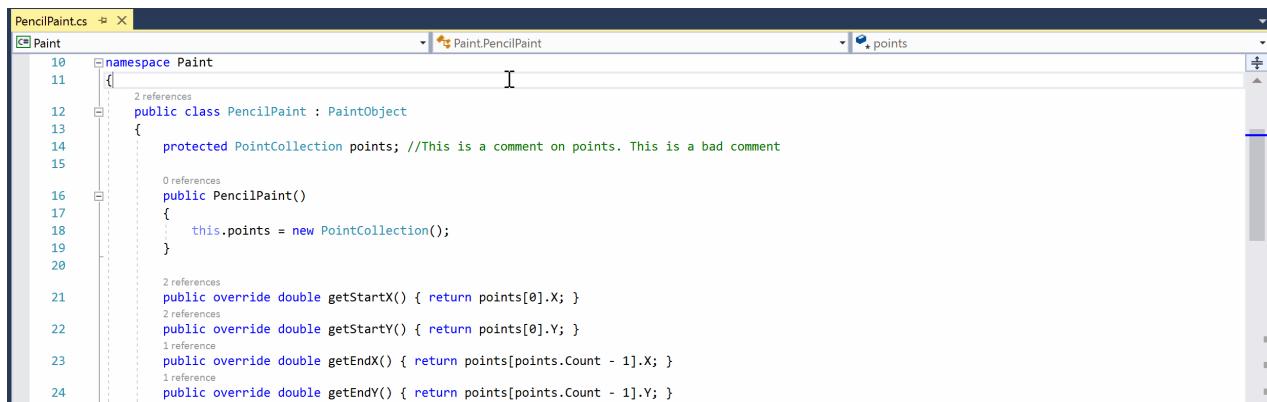
Developers often need to view the source code definitions for types or class members they use in their code. In Visual Studio, the Go To Definition and Peek Definition features enable you to easily view the definition of a type or member. If the source code is not available, metadata is displayed instead.

Go To Definition

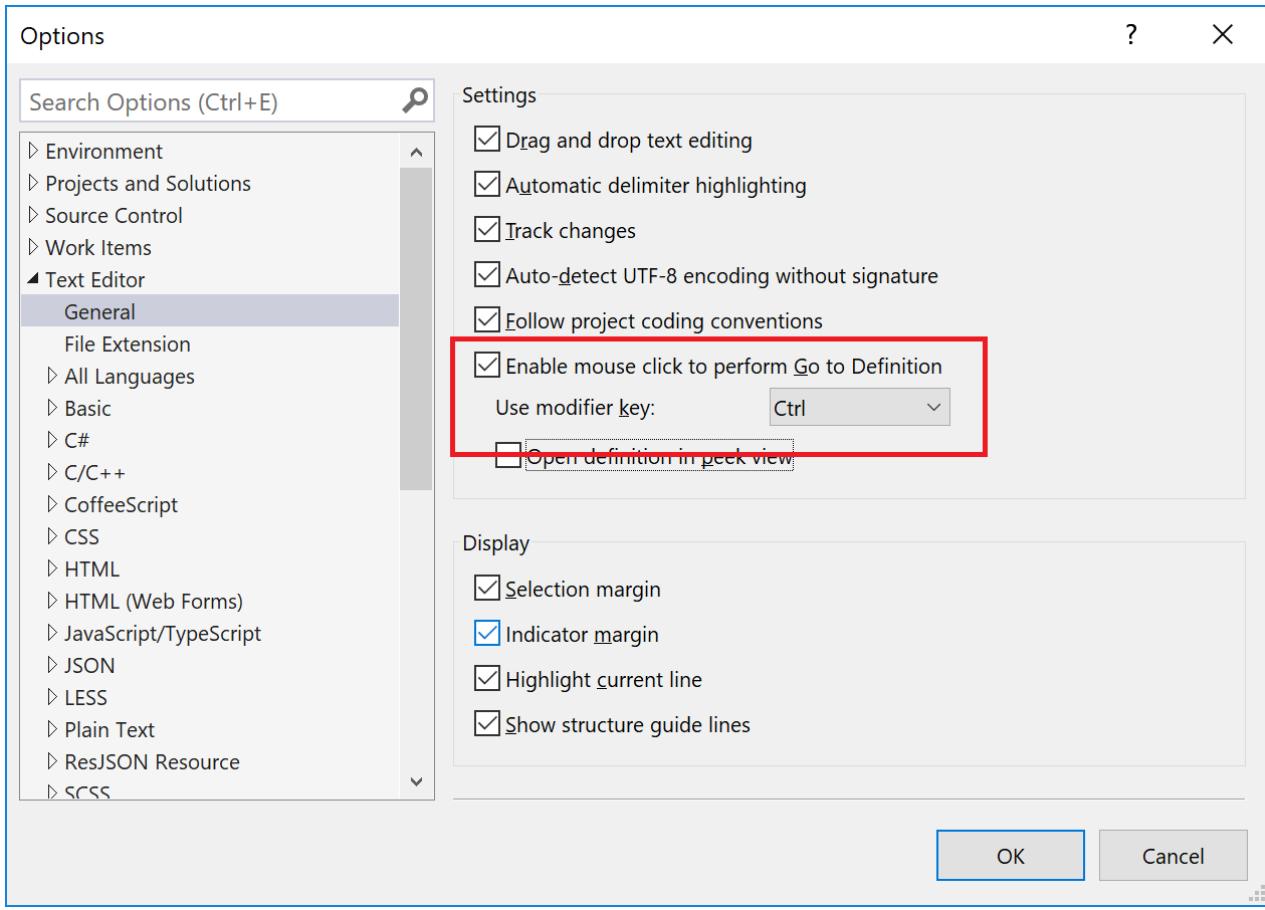
The Go To Definition feature navigates to the source of a type or member, and opens the result in a new tab. If you are a keyboard user, place your text cursor somewhere inside the symbol name and press **F12**. If you are a mouse user, either select **Go To Definition** from the context menu or use the **Ctrl-click** functionality described in the following section.

Ctrl-click Go To Definition

In Visual Studio 2017 version 15.4, there's an easier way for mouse users to quickly access Go To Definition. Symbols become clickable when you press **Ctrl** and hover over the type or member. To quickly navigate to the definition of a symbol, press the **Ctrl** key and then click on it. It's that easy!

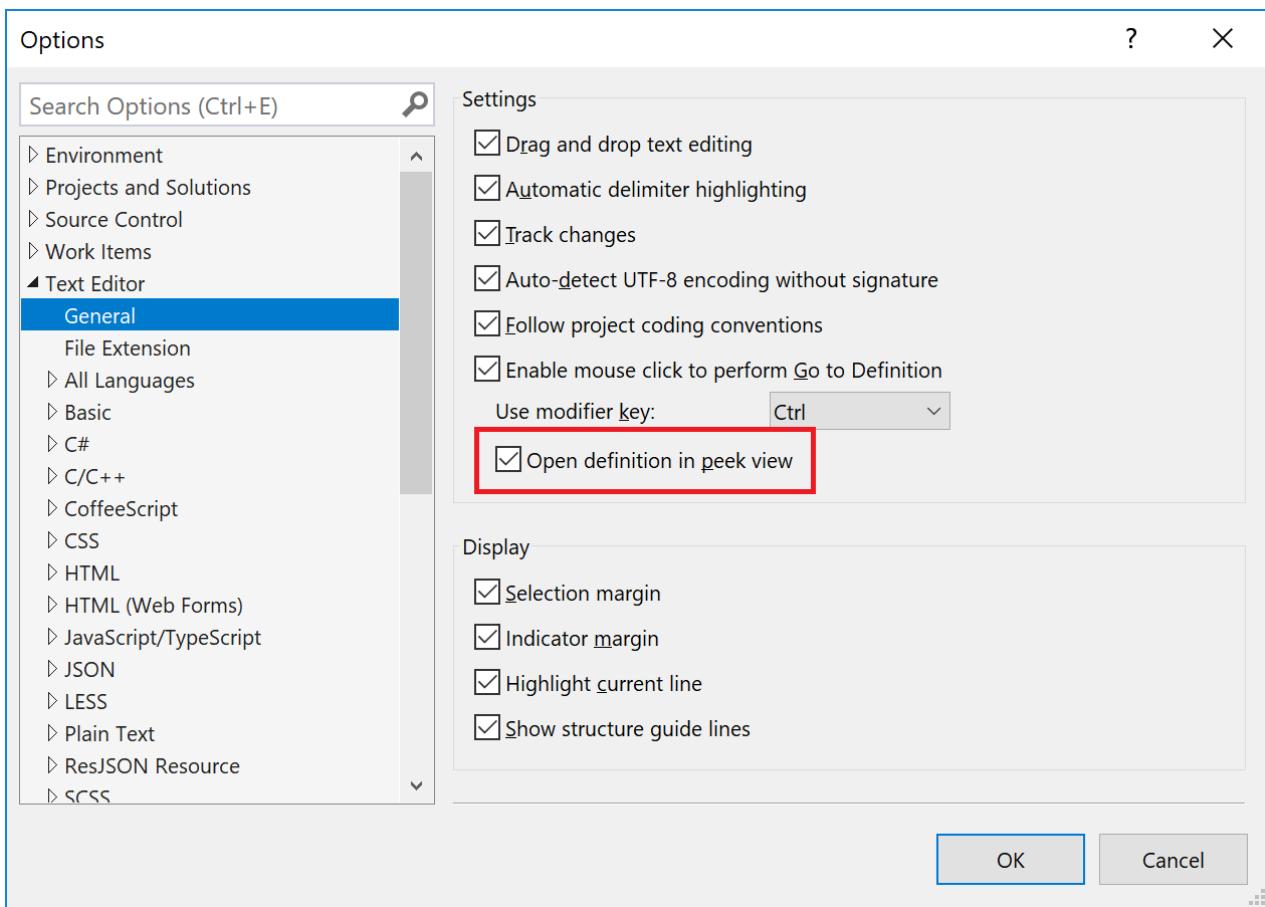


You can change the modifier key for mouse-click **Go To Definition** by going to **Tools**, **Options**, **Text Editor**, **General**, and selecting either **Alt** or **Ctrl+Alt** from the **Use modifier key** drop-down. You can also disable mouse-click **Go To Definition** by unchecking the **Enable mouse click to perform Go To Definition** checkbox.

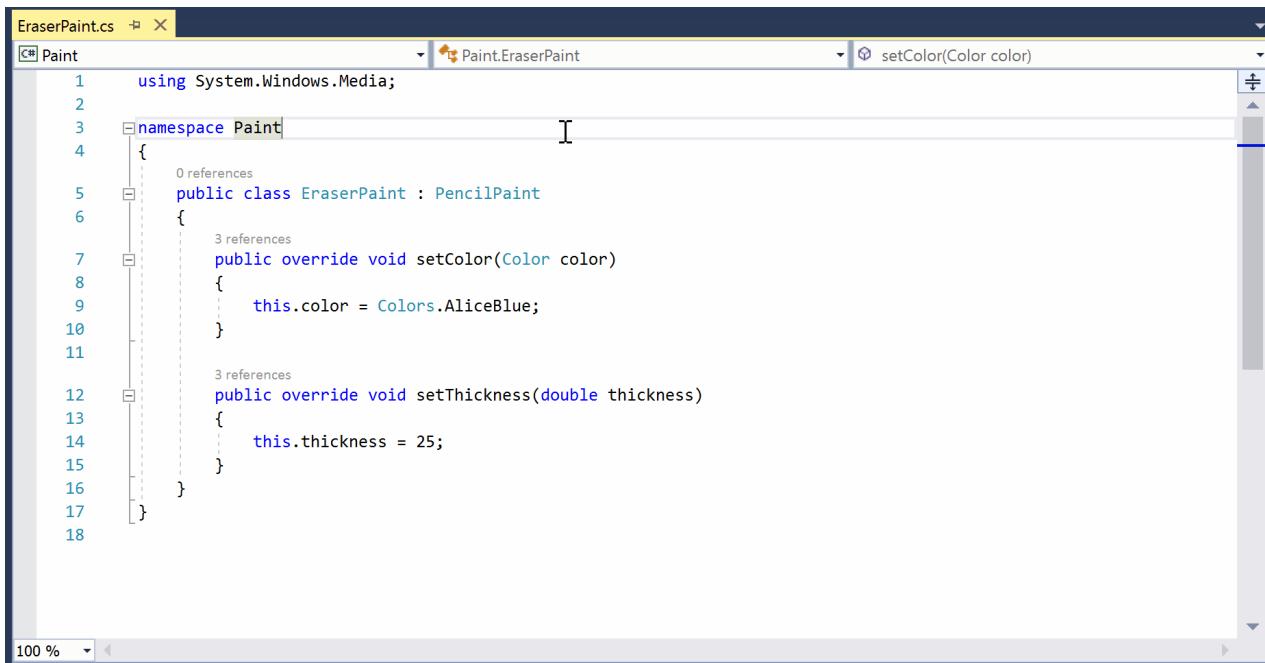


Peek Definition

The Peek Definition feature lets you preview the definition of a type without leaving your current location in the editor. If you are a keyboard user, place your text cursor somewhere inside the type or member name and press **Alt + F12**. If you are a mouse user, you can select **Peek Definition** from the context menu. In Visual Studio 2017 version 15.4 and later, there is a new way to peek view a definition by using the mouse. First, go to **Tools**, **Options**, **Text Editor**, **General**. Select the option **Open definition in peek view** and click **OK** to close the **Options** dialog box.



Then, press **Ctrl** (or whichever modifier key is selected in **Options**), and click on the type or member.



If you peek another definition from the popup window, you will start a breadcrumb path, which you can navigate using the circles and arrows that appear above the popup.

For more information, see [How to: View and Edit Code by Using Peek Definition \(Alt+F12\)](#).

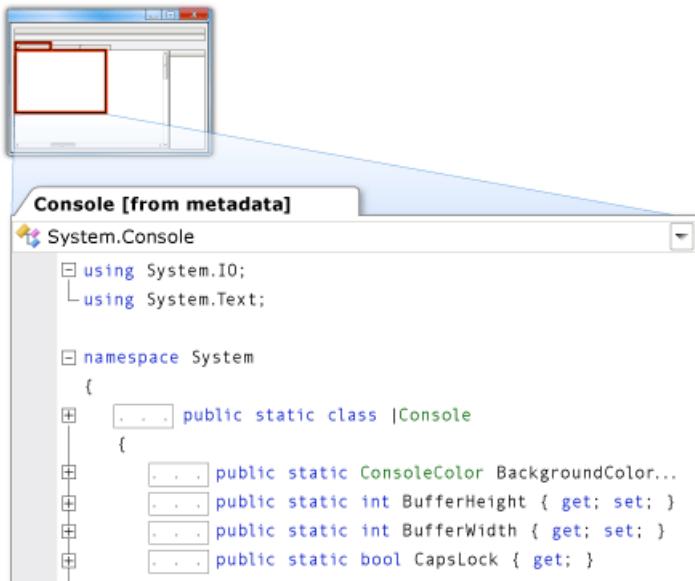
View metadata as source code (C#)

When you view the definition of C# types or members whose source code is not available, their metadata is displayed instead. You can see the declarations of the types and members, but not their implementations.

When you run the **Go To Definition** or **Peek Definition** command for an item whose source code is unavailable,

a tabbed document that contains a view of that item's metadata, displayed as source code, appears in the code editor. The name of the type, followed by **[from metadata]**, appears on the document's tab.

For example, if you run the **Go To Definition** command for [Console](#), metadata for [Console](#) appears in the code editor as C# source code. The code resembles its declaration, but does now show an implementation.



NOTE

When you try to run the **Go To Definition** or **Peek Definition** command for types or members that are marked as internal, Visual Studio does not display their metadata as source code, regardless of whether the referencing assembly is a friend or not.

View decompiled source definitions instead of metadata (C#)

New in **Visual Studio 2017 version 15.6**, you can set an option to see decompiled source code when you view the definition of a C# type or member who's source code is unavailable. To turn on this feature, choose **Tools > Options** from the menu bar. Then, expand **Text Editor > C# > Advanced**, and select **Enable navigation to decompiled sources**.

The screenshot shows a Windows application window titled "JsonSerializer [from metadata]". Inside, the code editor displays the following C# code:

```
using (StreamWriter sw = new StreamWriter(@"C:\json.txt"))
using (JsonWriter writer = new JsonTextWriter(sw))
{
    serializer.Serialize(writer, cryptid);

    /// <param name="value">The <see cref="T:System.Object" /> to serialize.</param>
    public void Serialize(JsonWriter jsonWriter, object value)
    {
        this.SerializeInternal(jsonWriter, value, null);
    }

    internal virtual void SerializeInternal(JsonWriter jsonWriter, object value, Type objectType)
    {
        ValidationUtils.ArgumentNotNull(jsonWriter, "jsonWriter");
        Formatting? nullable = null;
        if (this._formatting.HasValue && jsonWriter.Formatting != this._formatting)
        {
            nullable = jsonWriter.Formatting;
            jsonWriter.Formatting = this._formatting.GetValueOrDefault();
        }
    }
}
```

NOTE

Visual Studio reconstructs method bodies using ILSpy decompilation. The first time you access this feature, you must agree to a legal disclaimer regarding software licensing and copyright and trademark laws.

See also

[Navigating Code How to: View and Edit Code by Using Peek Definition \(Alt+F12\)](#)

How to: View and edit code by using Peek Definition (Alt+F12)

1/13/2018 • 3 min to read • [Edit Online](#)

You can use the **Peek Definition** command to view and edit code without switching away from the code that you're writing. **Peek Definition** and **Go To Definition** show the same information, but **Peek Definition** shows it in a pop-up window, and **Go To Definition** shows the code in a separate code window. **Go To Definition** causes your context (that is, the active code window, current line, and cursor position) to switch to the definition code window. By using **Peek Definition**, you can view and edit the definition and move around inside the definition file while keeping your place in the original code file.

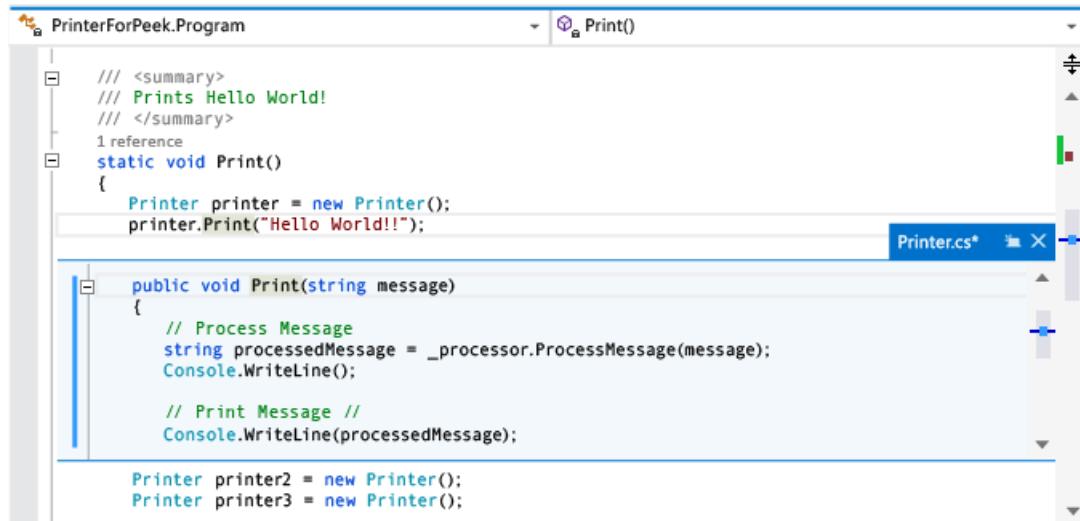
You can use **Peek Definition** with C#, Visual Basic, and C++ code. In Visual Basic, **Peek Definition** shows a link to the **Object Browser** for symbols that don't have definition metadata (for example, .NET Framework types that are built in).

Working with Peek Definition

To open a Peek Definition window

1. You can peek a definition by choosing **Peek Definition** from the context menu for a type or member that you want to explore. In Visual Studio 2017 version 15.4 and later, if the option is enabled, you can also peek a definition using the mouse, by pressing **Ctrl** (or another modifier) and clicking the member name. Or, from the keyboard, press **Alt+F12**.

This illustration shows the **Peek Definition** window for a method that's named `Print()`:



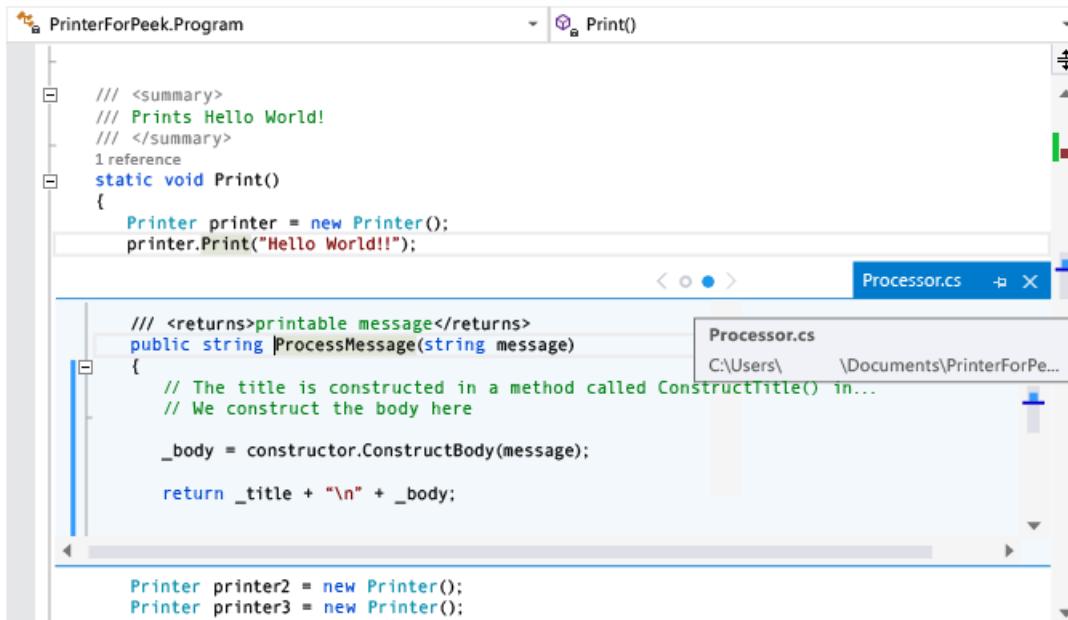
The definition window appears below the `printer.Print("Hello World!")` line in the original file. The window doesn't hide any of the code in your original file. The lines that follow `printer.Print("Hello World!")` appear under the definition window.

2. You can move the cursor to different locations in the peek definition window. You can also still move around in the original code window.
3. You can copy a string from the definition window and paste it in the original code. You can also drag and drop the string from the definition window to the original code without deleting it from the definition window.

4. You can close the definition window by choosing the **Esc** key or the **Close** button on the definition window tab.

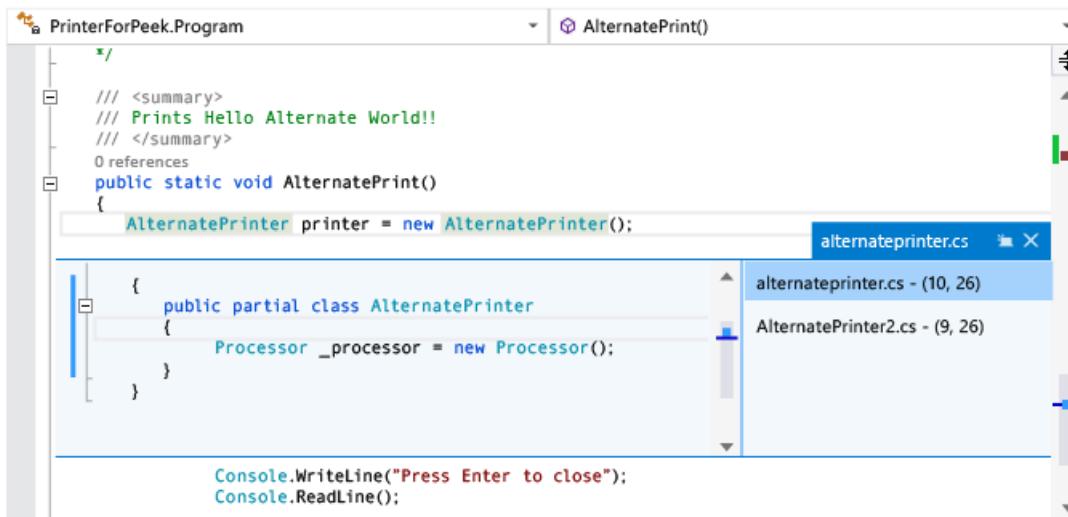
Open a Peek Definition window from within a Peek Definition window

If you already have a **Peek Definition** window open, you can call **Peek Definition** again on the code in that window. Another definition window opens. A set of breadcrumb dots appears next to the definition window tab, which you can use to navigate between definition windows. The tooltip on each dot shows the file name and path of the definition file that the dot represents.



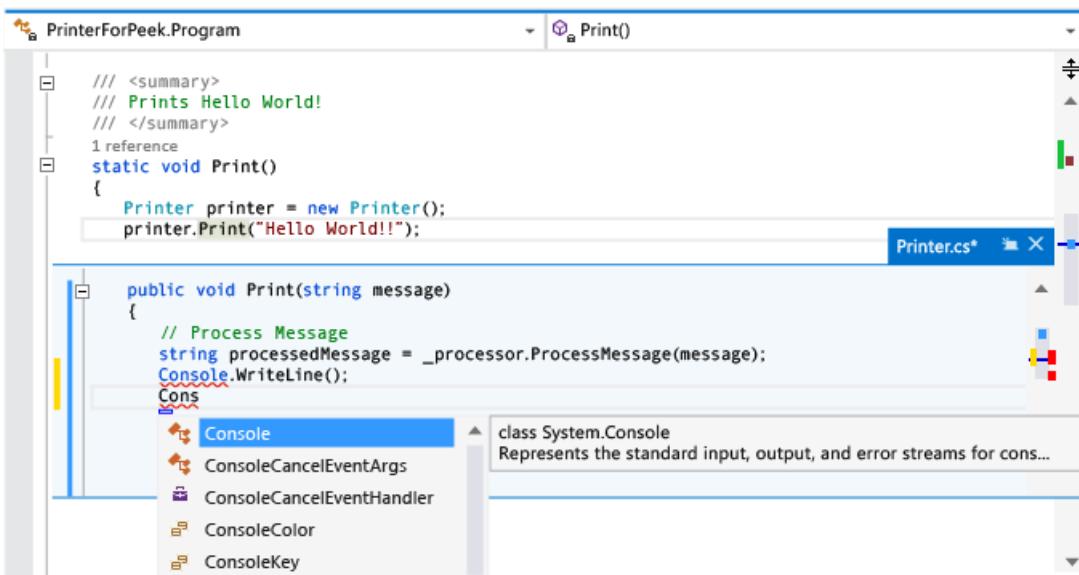
Peek Definition with multiple results

If you use **Peek Definition** on code that has more than one definition (for example, a partial class), a result list appears to the right of the code definition view. You can choose any result in the list to display its definition.



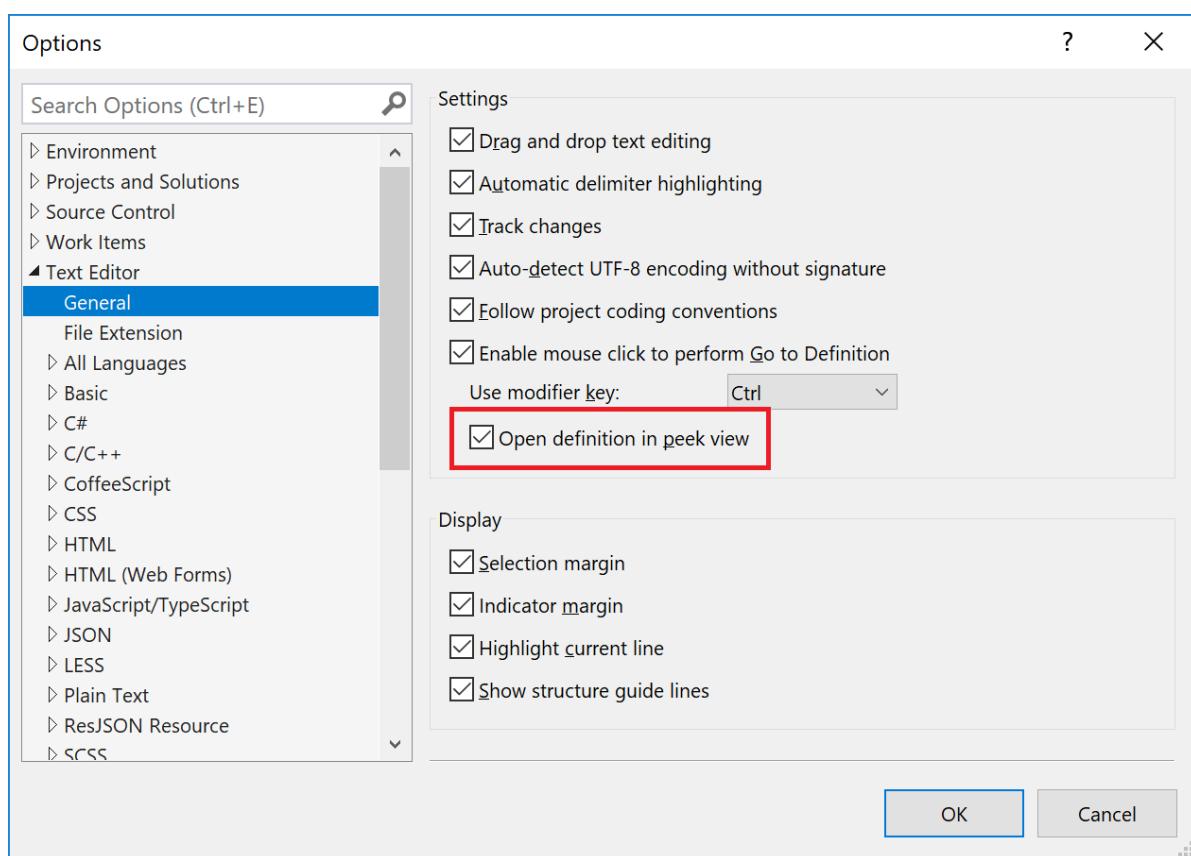
Edit inside the Peek Definition window

When you start to edit inside a **Peek Definition** window, the file that you're modifying automatically opens as a separate tab in the code editor and reflects the changes that you've made. You can continue to make, undo, and save changes in the **Peek Definition** window, and the tab will continue to reflect those changes. Even if you close the **Peek Definition** window without saving your changes, you can make, undo, and save more changes in the tab, picking up exactly where you left off in the **Peek Definition** window.



To change options for Peek Definition

1. Go to **Tools > Options > Text Editor > General**.
2. Select the option **Open definition in peek view**.
3. Click **OK** to close the **Options** dialog box.



Keyboard shortcuts for Peek Definition

You can use these keyboard shortcuts with the **Peek Definition** window:

| FUNCTIONALITY | KEYBOARD SHORTCUT |
|-----------------------------|-------------------|
| Open the definition window | Alt+F12 |
| Close the definition window | Esc |

| FUNCTIONALITY | KEYBOARD SHORTCUT |
|---|---------------------------|
| Promote the definition window to a regular document tab | Shift+Alt+Home |
| Navigate between definition windows | Ctrl+Alt+- and Ctrl+Alt+= |
| Navigate between multiple results | F8 and Shift+F8 |
| Toggle between the code editor window and the definition window | Shift+Esc |

NOTE

You can also use the same keyboard shortcuts to edit code in a **Peek Definition** window as you use elsewhere in Visual Studio.

See also

[Navigating Code](#)

[Go To Definition and Peek Definition](#)

[Productivity Tips](#)

Find code using Go To commands

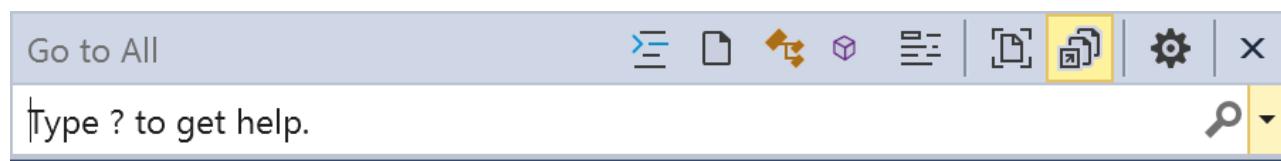
12/22/2017 • 2 min to read • [Edit Online](#)

Visual Studio's **Go To** commands perform a focused search of your code to help you quickly find specified items. You can go to a specific line, type, symbol, file, and member from a simple, unified interface. This feature exists in Visual Studio 2017 and later.

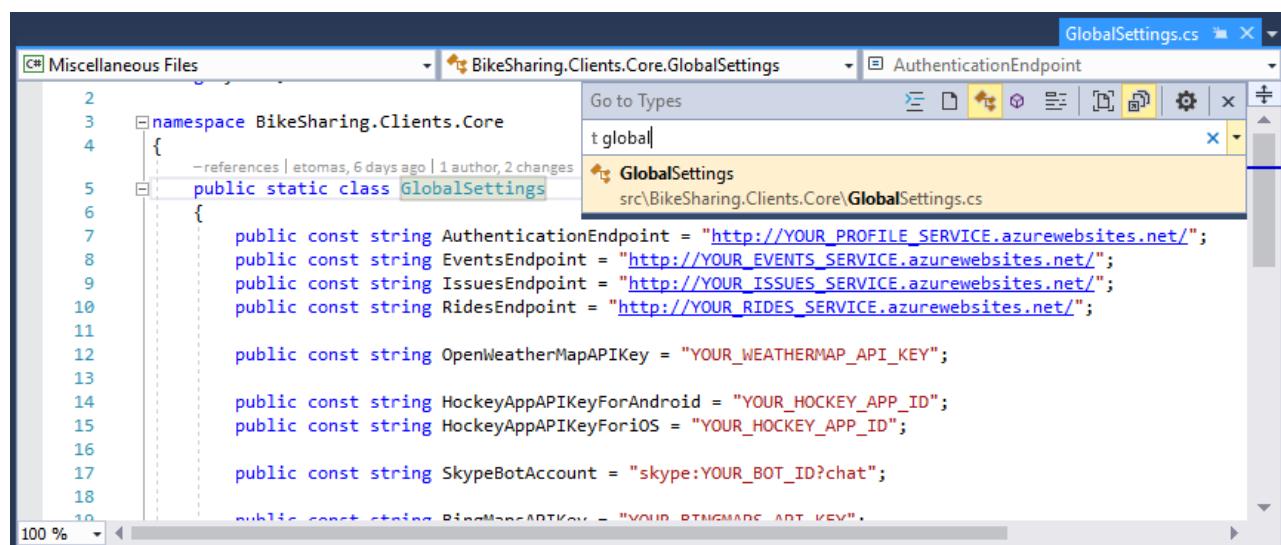
How to use it

| INPUT | FUNCTION |
|-----------------|--|
| Keyboard | Press Ctrl + t or Ctrl + , |
| Mouse | Select Edit, Go To, Go To All |

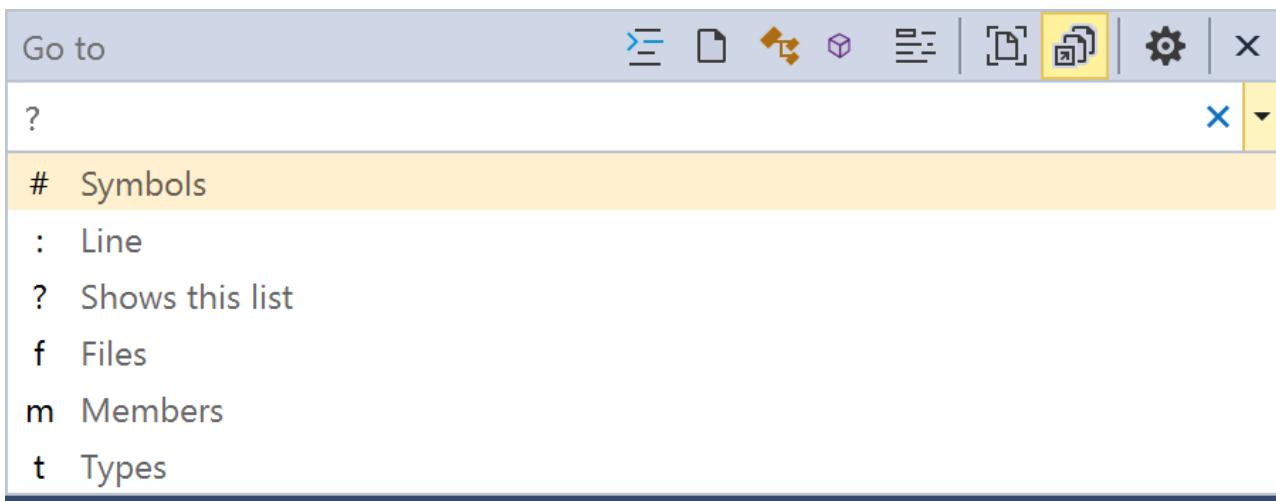
This will display a small window at the top right of your code editor, by default.



As you type in the text box, the results appear in a drop-down list below the text box. To go to an element, choose it in the list.

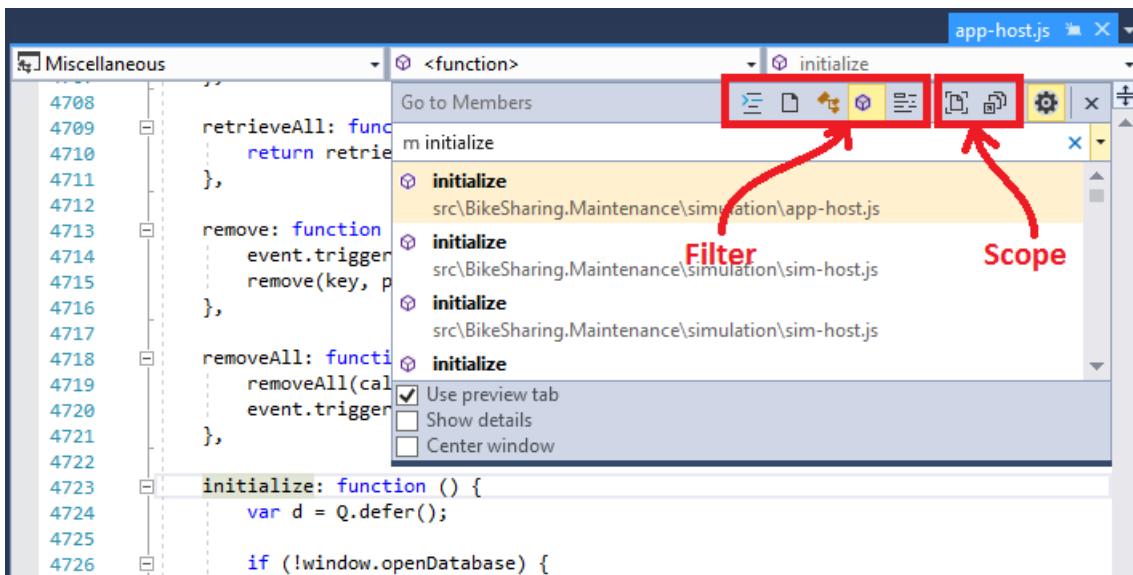


You can also enter a question mark (?) to get additional help.



Filtered Searches

By default, the specified item is searched for in all solution items. However, you can limit your code search to specific element types by prefacing the search terms with certain characters. You can also quickly change the search filter by choosing buttons on the Go To dialog box toolbar. Buttons that change the type filters are on the left side, and buttons that change the scope of the search are on the right side.



Filter to a specific type of code element

To narrow your search to a specific type of code element, you can either specify a prefix in the search box, or select one of the five filter icons:

| PREFIX | ICON | SHORTCUT | DESCRIPTION |
|--------|------|----------------|---------------------------------|
| # | 怄 | Ctrl+1, Ctrl+S | Go to the specified symbol |
| f | 📁 | Ctrl+1, Ctrl+F | Go to the specified file |
| m | ≡ | Ctrl+1, Ctrl+M | Go to the specified member |
| t | ⚡ | Ctrl+1, Ctrl+T | Go to the specified type |
| : | ☰ | Ctrl+G | Go to the specified line number |

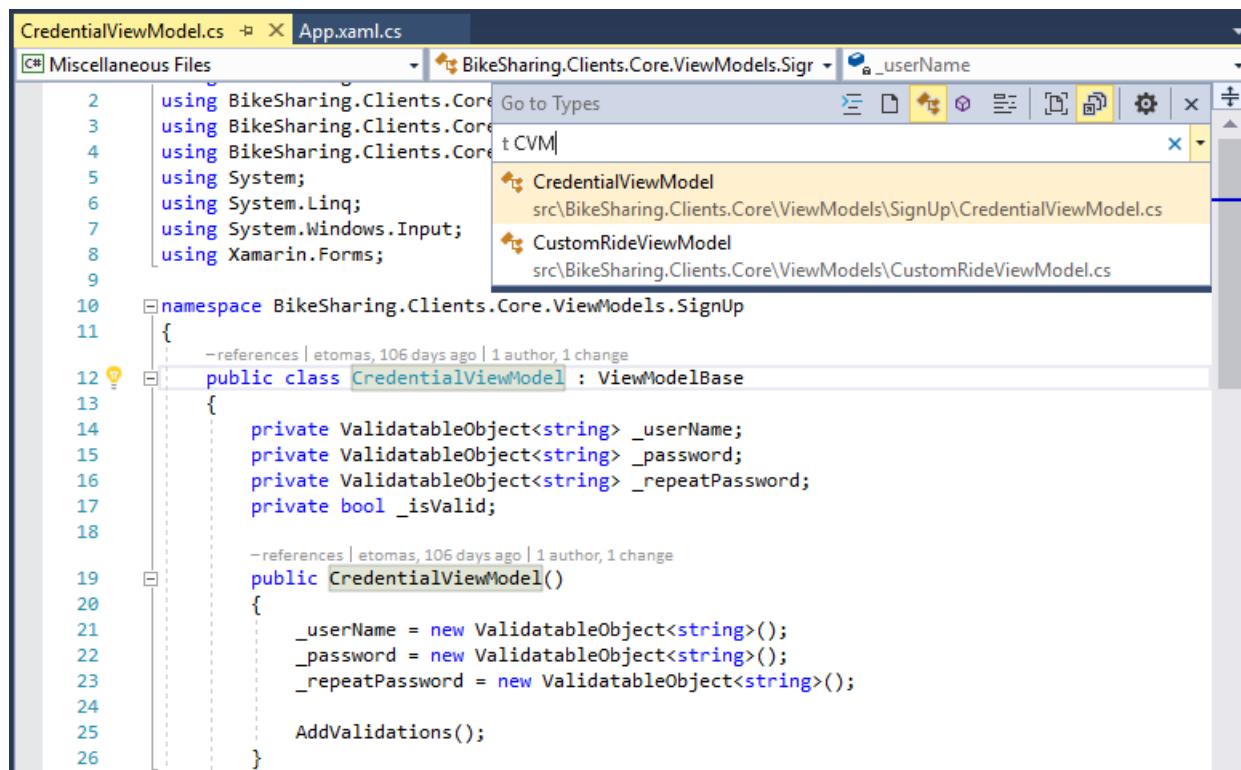
Filter to a specific location

To narrow your search to a specific location, select one of the two document icons:

| ICON | DESCRIPTION |
|------|--|
| | Search current document only |
| | Search external documents in addition to those located in the project/solution |

Camel casing

If you use [camel casing](#) in your code, you can find code elements faster by entering only the capital letters of the code element name. For example, if your code has a type called `CredentialViewModel`, you can narrow down the search by choosing the Type filter ('t') and then entering just the capital letters of the name (`CVM`) in the Go To dialog box. This feature can be helpful if your code has long names.



Settings

Selecting the gear icon lets you change how this feature works:

| SETTING | DESCRIPTION |
|-----------------|--|
| Use preview tab | Display the selected item immediately in the IDE's preview tab |
| Show details | Display project, file, line, and summary information from documentation comments in the window |
| Center window | Move this window to the top-center of the code editor, instead of the top-right |

See Also

[Navigating Code](#)

[Go To Definition](#) and [Peek Definition](#)

Customizing the Editor

1/23/2018 • 1 min to read • [Edit Online](#)

You can customize the formatting, tabs, fonts and colors, IntelliSense, and many other features of the Text Editor in general, or for a specific programming language. The topics in this section show you how to perform some of these customizations.

For more information about the Text Editor options, see [Text Editor Options Dialog Box](#).

See also

[Quickstart: Personalize the Visual Studio IDE and Editor](#)

[Writing Code](#)

[Setting Bookmarks in Code](#)

[General, Environment, Options Dialog Box](#)

[Documents, Environment, Options Dialog Box](#)

[Fonts and Colors, Environment, Options Dialog Box](#)

How to: change text case in the Editor

12/22/2017 • 1 min to read • [Edit Online](#)

You can use menu commands or keyboard shortcuts to convert the case of selected text to all upper case or to all lower case.

NOTE

The dialog boxes and menu commands you see might differ from those described in this article, depending on your active settings or edition. To change your settings, for example to **General** or **Visual C++** settings, choose **Tools, Import and Export Settings**, and then choose **Reset all settings**.

To switch text to upper case

1. Select the text you want to convert.
2. On the **Edit** menu, select **Advanced**.
3. To convert text to all upper case, choose **Make Uppercase**, or press **Ctrl+Shift+U**.

— or —

To convert text to all lower case, choose **Make Lowercase**, or press **Ctrl+U**.

TIP

To revert to the previous case formatting before this change, select **Undo** from the **Edit** menu.

See also

[Customizing the Editor](#)

[Text Editor Options Dialog Box](#)

[Writing Code](#)

How to: manage editor modes

12/22/2017 • 1 min to read • [Edit Online](#)

You can display the Visual Studio code editor in various display modes.

NOTE

The dialog boxes and menu commands you see might differ from those described in this article depending on your active settings or edition. To change your settings, for example to **General** or **Visual C++** settings, choose **Tools, Import and Export Settings**, and then choose **Reset all settings**.

Enabling Full Screen mode

You can choose to hide all tool windows and view only document windows by enabling **Full Screen** mode.

To enable Full Screen mode

- Press **Alt+Shift+Enter** to enter or exit **Full Screen** mode.
-- or --
- Issue the command `View.Fullscreen` in the **Command** window.

Enabling Virtual Space mode

In **Virtual Space** mode, spaces are inserted at the end of each line of code. Select this option to position comments at a consistent point next to your code.

To enable Virtual Space mode

1. Select **Options** from the **Tools** menu.
2. Expand the **Text Editor** folder, and choose **All Languages** to set this option globally, or choose a specific language folder. For example, to turn on line numbers only in Visual Basic, choose the Basic, Text Editor node.
3. Select **General** options, and under **Settings**, select **Enable Virtual Space**.

NOTE

Virtual Space is enabled in **Column Selection** mode. When **Virtual Space** mode is not enabled, the insertion point moves from the end of one line directly to the first character of the next.

See also

[Customizing the Editor](#)

[Customize window layouts in Visual Studio](#)

[Fonts and Colors, Environment, Options Dialog Box](#)

How to: Manage Editor Windows

12/22/2017 • 1 min to read • [Edit Online](#)

You can work on code in several locations at once. Do this by splitting an Editor window, or by opening several instances of editor windows.

NOTE

Not all editor windows support multiple instances.

Splitting an editor window

An instance of an editor window can be split into two separate views for easier editing.

To split a pane

1. Click within the editor window to give it focus.
2. From the **Window** menu, select **Split**.

The editing area divides into two panes separated by a splitter bar. You can scroll these panes independently to view and edit different parts of the active document at the same time. Any changes made in one pane are reflected in the other.

TIP

To make one pane larger than the other, drag the splitter bar upward or downward.

To return to single-pane view

- From the **Window** menu, select **Remove Split**.

Creating New Windows

You can also create multiple instances of an editor window. This feature allows you to open a lengthy document in more than one instance of an editor, so that you can view and edit different sections simultaneously in separate, full-sized editor windows.

To create a new window

- On the **Window** menu, click **New Window**.

A new tabbed instance of the editor is added.

See Also

[Customizing the Editor](#)

[Writing Code](#)

[Customizing window layouts](#)

How to: Change Fonts and Colors in the Editor

12/22/2017 • 1 min to read • [Edit Online](#)

You can change the default font face, adjust the font size, and change the foreground and background colors for various text **Display items** in the Code Editor. When changing font settings keep in mind the following information:

- The settings for **Font** and **Size** are global for all text elements in all Visual Studio editors.
- The names of fixed width fonts are listed in bold.
- **Item foreground**, **Item background**, and **Bold** options can be set for each type of text element. For example, if you change colors and select **Bold** for **Comment** and **Bookmarks**, other types of text elements will be unaffected.

NOTE

The dialog boxes and menu commands you see might differ from those described in **Help** depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

To change the default font face, size, and colors

1. Select **Options** from the **Tools** menu, and in the **Environment** folder, select **Fonts and Colors**.

The [Fonts and Colors, Environment, Options Dialog Box](#) opens.

2. In **Show settings for**, select **Text Editor**.
3. Modify the **Font** and **Size** options to change the font face and size for all text elements in all editors.
4. Select the appropriate item in **Display items**, and then modify the **Item foreground** and **Item background** options.

TIP

Click **Use Defaults** to reset the default settings.

5. Click **OK**.

See Also

[Customizing the Editor](#)

[Text Editor Options Dialog Box](#)

[Writing Code](#)

[How to: Change Fonts and Colors](#)

How to: Manage Word Wrap in the Editor

1/26/2018 • 1 min to read • [Edit Online](#)

You can set and clear the **Word wrap** option. When this option is set, the portion of a long line that extends beyond the current width of the Code Editor window is displayed on the next line. When this option is cleared, for example, to facilitate the use of line numbering, you can scroll to the right to see the ends of long lines.

NOTE

The dialog boxes and menu commands you see might differ from those described in **Help** depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Procedure

To set word wrap preferences

1. On the **Tools** menu, select **Options**.
2. In the **Text Editor** folder, choose the **General** options in the **All Languages** subfolder to set this option globally.

— or —

Choose the **General** options in the subfolder for the language in which you are programming.

3. Under **Settings**, select or clear the **Word wrap** option.

When the **Word wrap** option is selected, the **Show visual glyphs for word wrap** option is enabled.

4. Select the **Show visual glyphs for Word Wrap** option if you prefer to display a return-arrow indicator where a long line wraps onto a second line. Clear this option if you prefer not to display indicator arrows.

NOTE

These reminder arrows are not added to your code: they are for display purposes only.

See also

[Customizing the Editor](#)

[Text Editor Options Dialog Box](#)

[Writing Code](#)

How to: Display Line Numbers in the Editor

12/22/2017 • 1 min to read • [Edit Online](#)

You can display or hide line numbering in your code.

NOTE

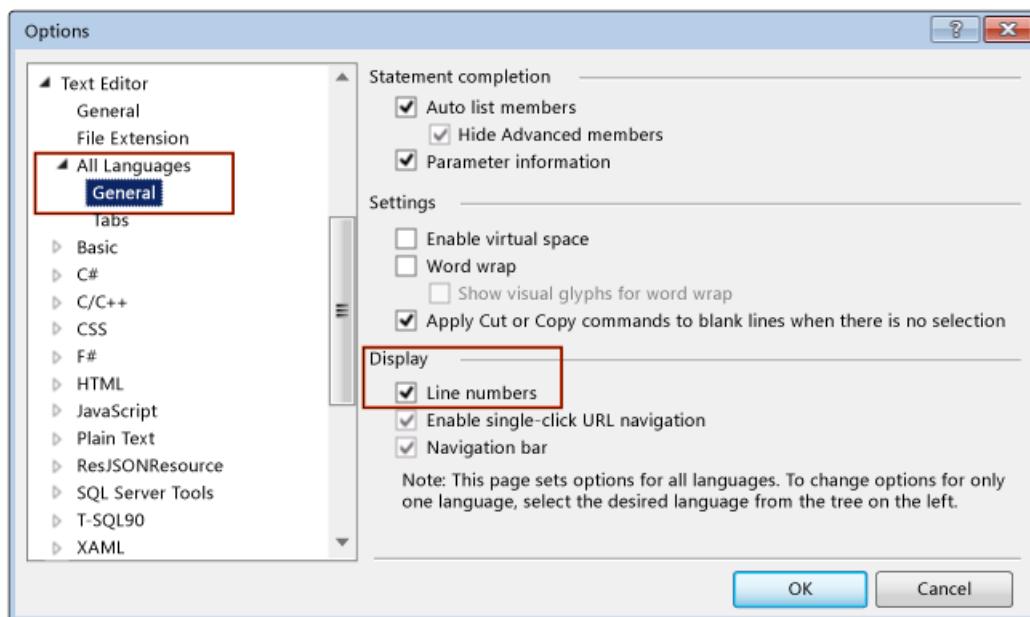
Depending on your active settings, the dialog boxes and menu commands that appear might differ from the ones that are described here. To change your settings, open **Tools / Import and Export Settings**. For more information, see [Personalize the Visual Studio IDE](#).

Display Line Numbers in Code

Line numbers aren't added to your code; they're just for reference. If you want line numbers to print, in the **Print** dialog box, select the **Include line numbers** check box.

To display line numbers in code

1. On the menu bar, choose **Tools, Options**. Expand the **Text Editor** node, and then select either the node for the language you are using, or **All Languages** to turn on line numbers in all languages. Or, you can type **line number** in the **Quick Launch** box.
2. Select the **Line numbers** checkbox.



NOTE

Depending on your language or settings, you may have to select the **Show All Settings** check box in the **Options** dialog box to reveal the **All Languages** sub-node.

See Also

[Customizing the Editor](#)

[Text Editor Options Dialog Box](#)

[Writing Code](#)

How to: Display URLs as Links in the Editor

12/22/2017 • 1 min to read • [Edit Online](#)

You can choose to have the Code Editor treat Uniform Resource Locators (URLs) in your code as active links. When you use this feature, URLs:

- Appear underlined.
- Display a **ToolTip** when you hover over them.
- Attempt to open the Web site indicated when you **CTRL** + left-click on the link. By default, the Web site is displayed in the internal Web browser.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Procedure

To display URLs as links

1. On the **Tools** menu, click **Options**.
2. Click **Text Editor**.
3. To change the option for only one language, expand the folder for that language and choose **General**.
—or—
To change the option for all languages, expand the **All Languages** folder and choose **General**.
4. Under **Display**, select **Enable single-click URL navigation**.

See Also

[Customizing the Editor](#)

[Text Editor Options Dialog Box](#)

[Writing Code](#)

Setting language-specific editor options

12/22/2017 • 1 min to read • [Edit Online](#)

Visual Studio offers a variety of text editor options that apply to specific programming languages. You can configure options in the **Options** dialog box, which is accessed from the **Tools** menu. You can also configure some editor settings on a project- or codebase-basis by creating an EditorConfig file. See [Create portable, custom editor settings with EditorConfig](#).

Settings available in the Options dialog box

[Options, Text Editor, Basic \(Visual Basic\)](#)

Describes settings for end constructs, code reformatting, outlining, and error correction suggestions, among others, for Visual Basic code.

[Options, Text Editor, C/C++, Formatting](#)

Describes outlining, indenting, Quick Info, and other settings for C and C++ code.

[Options, Text Editor, C/C++, Advanced](#)

Describes settings for IntelliSense and database files when using C and C++.

[Options, Text Editor, C#, Formatting](#)

Describes settings for indenting, new line formatting, and wrapping text, among others, for C#.

[Options, Text Editor, C#, Advanced](#)

Describes outlining, error identification, and XML documentation comment settings for C#.

[Options, Text Editor, C#, IntelliSense](#)

Describes settings that specify how the IntelliSense completion list behaves when you work in C# code.

[Options, Text Editor, XAML, Formatting](#)

Describes settings for element and attribute arrangement in XAML documents.

See also

[Customizing the Editor](#)

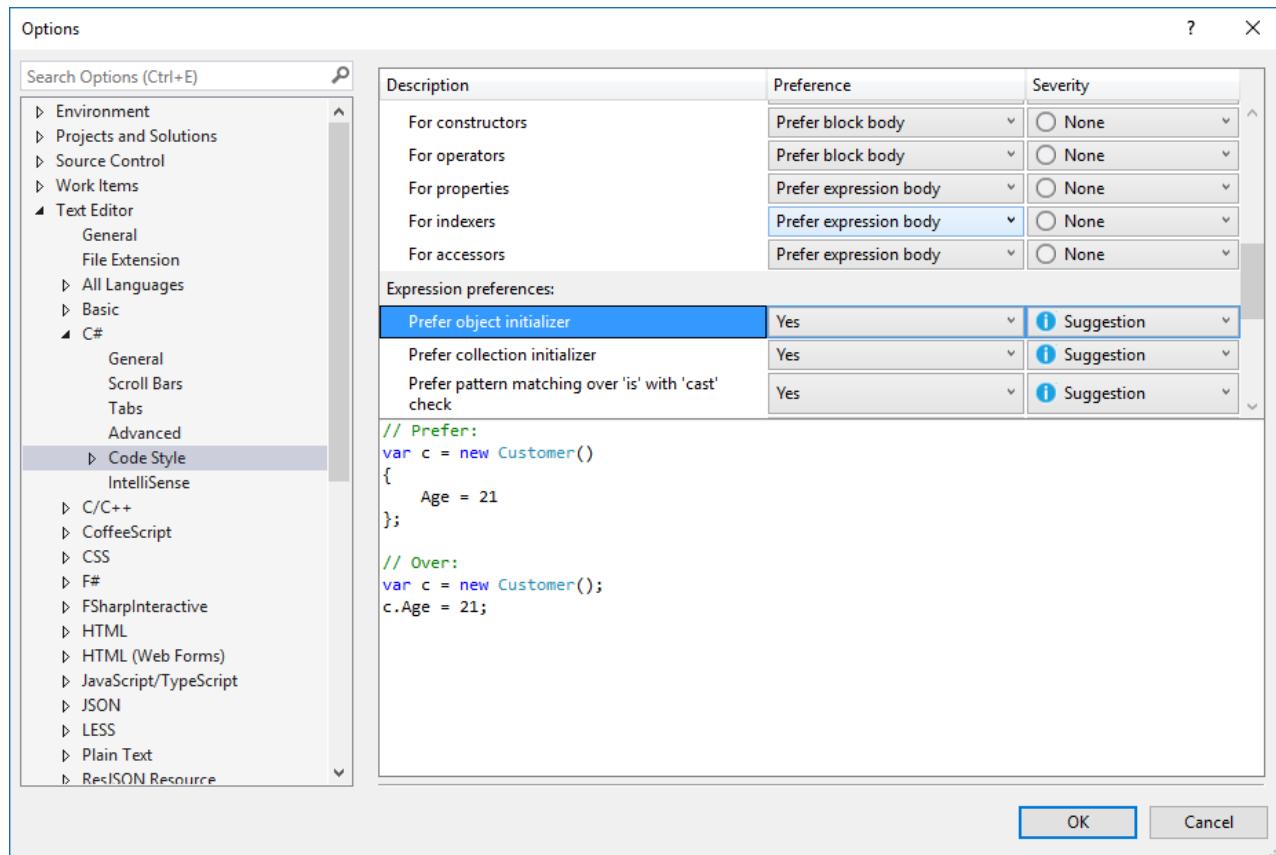
[Create portable, custom editor settings with EditorConfig](#)

[Personalize the Visual Studio IDE](#)—Provides links to topics that explain how to customize your settings, key bindings, and other features of the integrated development environment (IDE).

Code style preferences

1/13/2018 • 1 min to read • [Edit Online](#)

Code style preferences can be set for your C# and Visual Basic projects by opening the **Options** dialog box from the **Tools** menu. Select **Text Editor > C# or Basic > Code Style > General**. Options set in this window are applicable to the local machine. Each item in the list will show a preview of the preference when selected, as shown below.



For each item, you can set the **Preference** and **Severity** values using the drop downs on each line. Severity can be set to **None**, **Suggestion**, **Warning**, or **Error**. If you want to enable **Quick Actions** for a code style, ensure that the **Severity** setting is set to something other than **None**. The Quick Actions light bulb icon will appear when a non-preferred style is used, and you can choose an option on the Quick Actions list to automatically rewrite code to the preferred style.

Code Style settings for .NET can also be managed with an [EditorConfig](#) file. In this case, the settings in the EditorConfig file take precedence over options selected in the **Options** dialog box. You can use an EditorConfig file to enforce and configure the coding style for your entire repo or project.

See also

[Quick Actions](#)

[.NET coding convention settings for EditorConfig](#)

Create portable, custom editor settings with EditorConfig

2/6/2018 • 7 min to read • [Edit Online](#)

In Visual Studio 2017, you can add an [EditorConfig](#) file to your project or codebase to enforce consistent coding styles for everyone that works in the codebase. EditorConfig settings take precedence over global Visual Studio text editor settings. This means that you can tailor each codebase to use text editor settings that are specific to that project. You can still set your own personal editor preferences in the Visual Studio **Options** dialog box. Those settings apply whenever you're working in a codebase without an `.editorconfig` file, or when the `.editorconfig` file doesn't override a particular setting. An example of such a preference is indent style—tabs or spaces.

EditorConfig settings are supported by numerous code editors and IDEs, including Visual Studio. It's a portable component that travels with your code, and can enforce coding styles even outside of Visual Studio.

NOTE

When you add an EditorConfig file to your project in Visual Studio, the formatting of existing code is not changed unless you format the document (**Edit > Advanced > Format Document** or **Ctrl+K, Ctrl+D**). However, any new lines of code are formatted according to the EditorConfig settings.

Coding consistency

Settings in EditorConfig files enable you to maintain consistent coding styles and settings in a codebase, such as indent style, tab width, end of line characters, encoding, and more, regardless of the editor or IDE you use. For example, when coding in C#, if your codebase has a convention to prefer that indents always consist of five space characters, documents use UTF-8 encoding, and each line always ends with a CR/LF, you can configure an `.editorconfig` file to do that.

Coding conventions you use on your personal projects may differ from those used on your team's projects. For example, you might prefer that when you're coding, indenting adds a tab character. However, your team might prefer that indenting adds four space characters instead of a tab character. EditorConfig files resolve this problem by enabling you to have a configuration for each scenario.

Because the settings are contained in a file in the codebase, they travel along with that codebase. As long as you open the code file in an EditorConfig-compliant editor, the text editor settings are implemented. For more information about EditorConfig files, see the [EditorConfig.org](#) website.

Supported settings

The editor in Visual Studio supports the core set of [EditorConfig properties](#):

- `indent_style`
- `indent_size`
- `tab_width`
- `end_of_line`
- `charset`
- `trim_trailing_whitespace`
- `insert_final_newline`
- `root`

EditorConfig editor settings are supported in all Visual Studio-supported languages except for XML. In addition, EditorConfig supports [code style](#) and [naming](#) conventions for C# and Visual Basic.

Adding and removing EditorConfig files

Adding an EditorConfig file to your project or codebase does not convert existing styles to the new ones. For example, if you have indents in your file that are formatted with tabs, and you add an EditorConfig file that indents with spaces, the indent characters are not automatically converted to spaces. However, any new lines of code are formatted according to the EditorConfig file. Additionally, if you format the document (**Edit > Advanced > Format Document** or **Ctrl+K, Ctrl+D**), the settings in the EditorConfig file are applied to existing lines of code.

If you remove an EditorConfig file from your project or codebase, you must close and reopen any open code files to revert to the global editor settings for new lines of code.

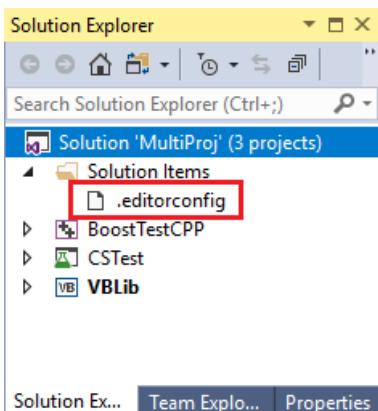
To add an EditorConfig file to a project or solution

1. Open a project or solution in Visual Studio. Select either the project or solution node, depending on whether your .editorconfig settings should apply to all projects in the solution or just one. You can also select a folder in your project or solution to add the .editorconfig file to.
2. From the menu bar, choose **Project > Add New Item...**, or press **Ctrl+Shift+A**.

The **Add New Item** dialog box opens.

3. In the categories on the left, choose **General**, and then choose the **Text File** template. In the **Name** text box, enter `.editorconfig` and then choose **Add**.

An .editorconfig file appears in Solution Explorer, and it opens in the editor.



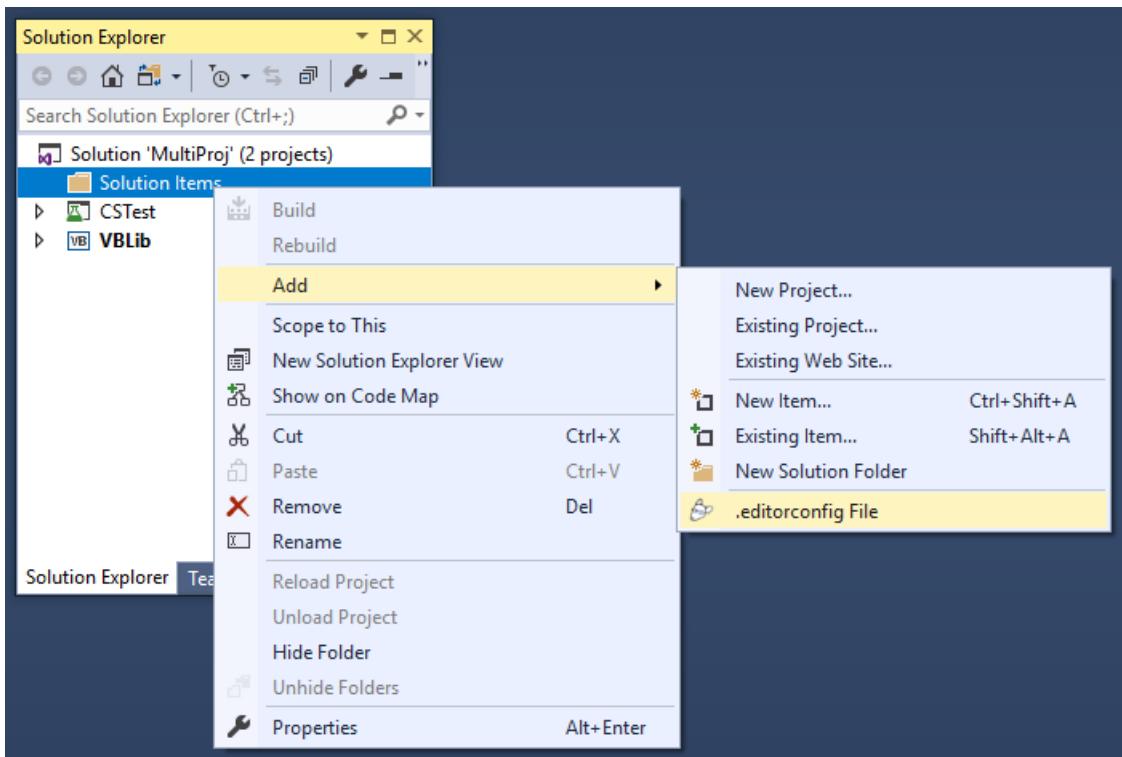
4. Edit the file as desired, for example:

```
root = true

[*.cs,vb]
indent_size = 4
trim_trailing_whitespace = true

[*.cs]
csharp_new_line_before_open_brace = methods
```

Alternatively, you can install the [EditorConfig Language Service extension](#). After you install this extension, simply choose **Add > .editorconfig File** from the right-click or context menu of the solution node, project node, or any folder in Solution Explorer.



Override EditorConfig settings

When you add an .editorconfig file to a folder in your file hierarchy, its settings apply to all applicable files at that level and below. You can also override EditorConfig settings for a particular project, codebase, or part of a codebase, such that it uses different conventions than other parts of the codebase. This can be useful when you incorporate code from somewhere else, and don't want to change its conventions.

To override some or all of the EditorConfig settings, add an .editorconfig file at the level of the file hierarchy you want those overridden settings to apply. The new EditorConfig file settings apply to files at the same level and any subdirectories.

```
.editorconfig
file1
file2
  .editorconfig
    file3
    file4
```

If you want to override some but not all of the settings, specify just those settings in the .editorconfig file. Only those properties that you explicitly list in the lower-level file are overridden. Other settings from higher-level .editorconfig files continue to apply. If you want to ensure that *no* settings from *any* higher-level .editorconfig files are applied to this part of the codebase, add the `root=true` property to the lower-level .editorconfig file:

```
# top-most EditorConfig file
root = true
```

EditorConfig files are read top to bottom, and the closest EditorConfig files are read last. Conventions from matching EditorConfig sections are applied in the order they were read, so conventions in closer files take precedence.

Editing EditorConfig files

Visual Studio helps you edit .editorconfig files by providing IntelliSense completion lists.

```
.editorconfig* ✘ X
1 root = true
2
3 [*.{cs,vb}]
4 indent_style = space
5
```

A screenshot of the Visual Studio code editor showing a .editorconfig file. The file contains five lines of configuration. The fourth line, 'indent_style = space', is selected and highlighted with a yellow background. A context menu is open over this line, displaying options: 'abc', 'cs', 'indent_style', 'root', 'space', 'true', and 'vb'. The 'cs' option is currently selected, indicated by a blue border around its text.

After you've edited your EditorConfig file, you must reload your code files for the new settings to take effect.

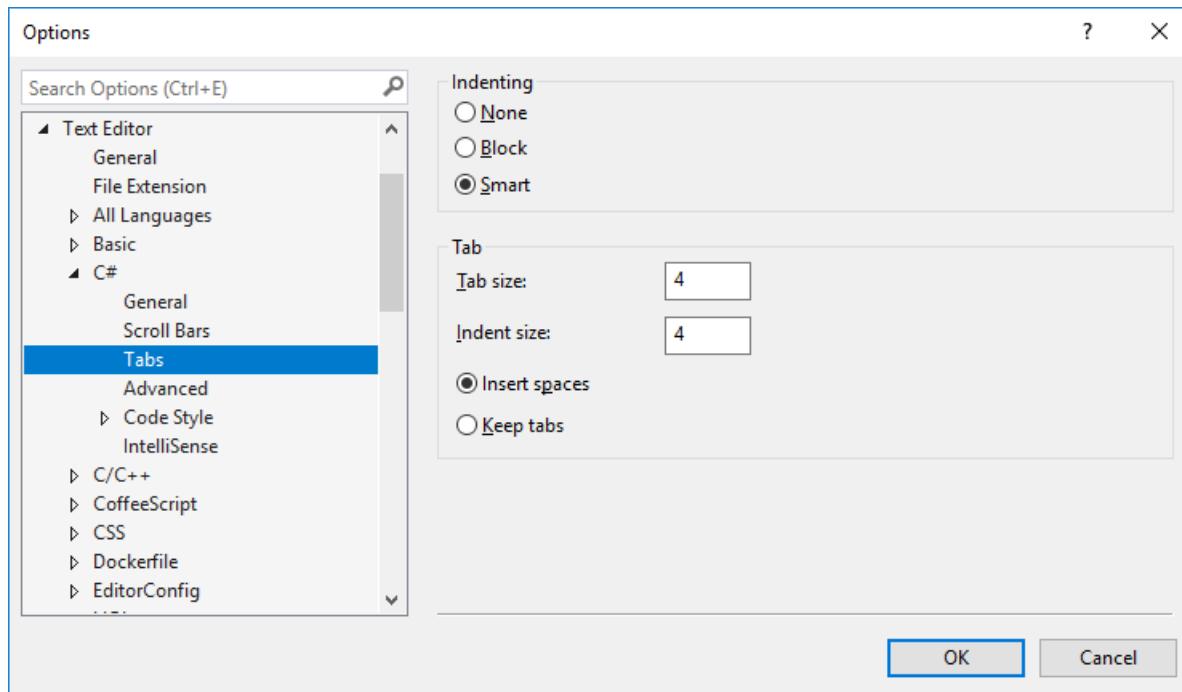
If you edit numerous .editorconfig files, you may find the [EditorConfig Language Service extension](#) helpful. Some of the features of this extension include syntax highlighting, improved IntelliSense, validation, and code formatting.

```
.editorconfig* ✘ X
.editorconfig (MultiProj)
1 root = true
2
3 [*.{cs,vb}]
4 indent_style = space
5
```

A screenshot of the Visual Studio code editor showing a .editorconfig file. The 'indent_style' setting is being typed, and an Intellisense tooltip is displayed. The tooltip shows 'Indent style' and 'Indentation style' as suggestions. Below the tooltip, a list of C# specific suggestions is shown, including 'indent_size', 'csharp_style_expression_bodied_indexers', 'csharp_style_inlined_variable_declaration', 'csharp_indent_case_contents', 'csharp_indent_labels', and 'csharp_indent_switch_labels'. The 'indent_size' suggestion is currently selected, indicated by a blue border around its text.

Example

The following example shows the indent state of a C# code snippet before and after adding an .editorconfig file to the project. The **Tabs** setting in the **Options** dialog box for the Visual Studio text editor is set to produce space characters when you press the **Tab** key.



As expected, pressing the **Tab** key on the next line indents the line by adding four additional white space

characters.



```
Program.cs
ConsoleApp1
ConsoleApp1.Program
Main(string[] args)

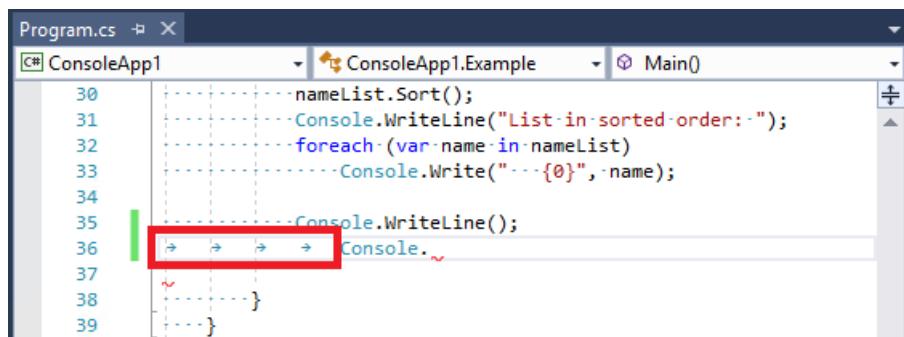
30     nameList.Sort();
31     Console.WriteLine("List in sorted order:");
32     foreach (var name in nameList)
33         Console.Write(" - {0}", name);
34
35     Console.WriteLine();
36     Console.~
37
38 }
39 }
```

Add a new file called `.editorconfig` to the project, with the following contents. The `[*.cs]` setting means that this change applies only to C# code files in the project.

```
# Top-most EditorConfig file
root = true

# Tab indentation
[*.cs]
indent_style = tab
```

Now, when you press the **Tab** key, you get tab characters instead of spaces.



```
Program.cs
ConsoleApp1
ConsoleApp1.Example
Main()

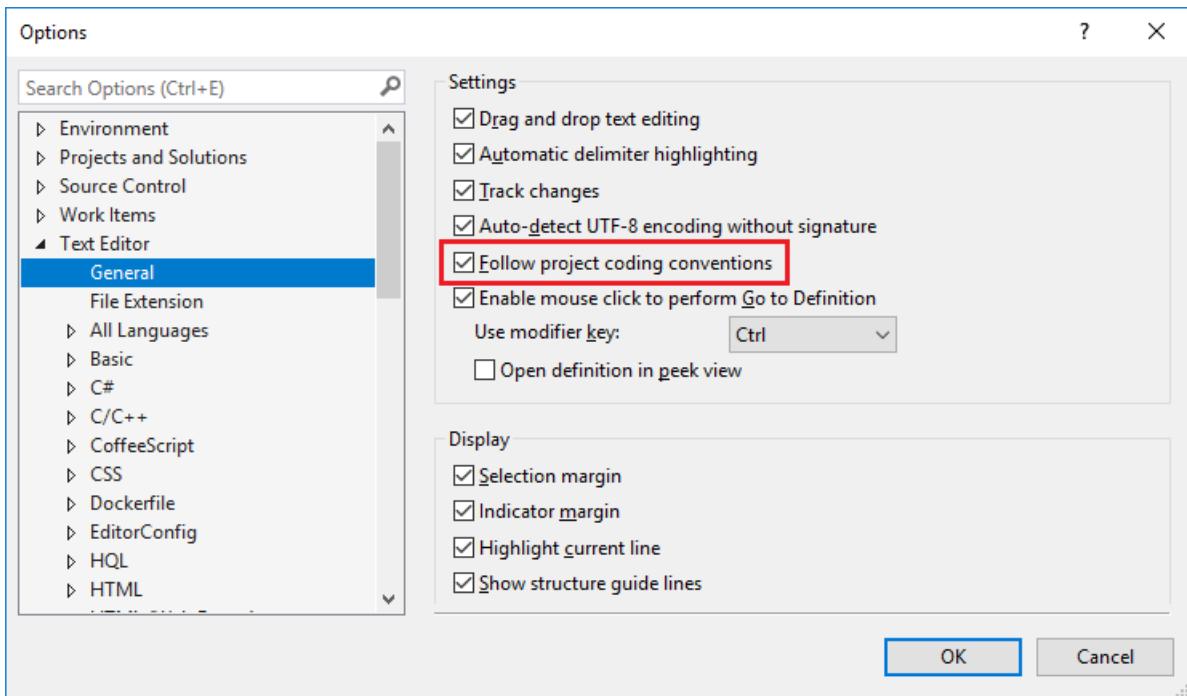
30     nameList.Sort();
31     Console.WriteLine("List in sorted order:");
32     foreach (var name in nameList)
33         Console.Write(" - {0}", name);
34
35     Console.WriteLine();
36     + + + + Console.~
37
38 }
39 }
```

Troubleshooting EditorConfig settings

If there is an `EditorConfig` file anywhere in the directory structure at or above your project's location, Visual Studio applies the editor settings in that file to your editor. In this case, you may see the following message in the status bar:

"User preferences for this file type are overridden by this project's coding conventions."

This means that if any editor settings in **Tools > Options > Text Editor** (such as indent size and style, tab size, or coding conventions) are specified in an `EditorConfig` file at or above the project in the directory structure, the conventions in the `EditorConfig` file override the settings in Options. You can control this behavior by toggling the **Follow project coding conventions** option in **Tools > Options > Text Editor**. Unchecking the option turns off `EditorConfig` support for Visual Studio.



You can find any .editorconfig files in parent directories by opening a command prompt and running the following command from the root of the disk that contains your project:

```
dir .editorconfig /s
```

You can control the scope of your EditorConfig conventions by setting the `root=true` property in the .editorconfig file at the root of your repo or in the directory that your project resides. Visual Studio looks for a file named .editorconfig in the directory of the opened file and in every parent directory. The search ends when it reaches the root filepath, or if an .editorconfig file with `root=true` is found.

See also

[.NET code style conventions](#)

[.NET naming conventions](#)

[Supporting EditorConfig for a language service](#)

[EditorConfig.org](#)

[Writing code in the editor](#)

.NET coding convention settings for EditorConfig

3/22/2018 • 30 min to read • [Edit Online](#)

In Visual Studio 2017, you can define and maintain consistent code style in your codebase with the use of an [EditorConfig](#) file. EditorConfig includes several core formatting properties, such as `indent_style` and `indent_size`. In Visual Studio, .NET coding conventions settings can also be configured using an EditorConfig file. EditorConfig files allow you to enable or disable individual .NET coding conventions, and to configure the degree to which you want the convention enforced via a severity level. To learn more about how to use EditorConfig to enforce consistency in your codebase, read [Create portable custom editor options](#). You can also look at the [.NET Compiler Platform's .editorconfig file](#) as an example.

There are three supported .NET coding convention categories:

- [Language Conventions](#)

Rules pertaining to the C# or Visual Basic language. For example, you can specify rules around using `var` or explicit types when defining variables, or preferring expression-bodied members.

- [Formatting Conventions](#)

Rules regarding the layout and structure of your code in order to make it easier to read. For example, you can specify rules around Allman braces, or preferring spaces in control blocks.

- [Naming Conventions](#)

Rules regarding the naming of code elements. For example, you can specify that `async` methods must end in "Async".

Language conventions

Rules for language conventions have the following format:

```
options_name = false|true : none|suggestion|warning|error
```

For each language convention rule, you must specify either **true** (prefer this style) or **false** (do not prefer this style), and a **severity**. The severity specifies the level of enforcement for that style.

The following table lists the possible severity values and their effects:

| SEVERITY | EFFECT |
|----------------|--|
| none or silent | Do not show anything to the user when this rule is violated. Code generation features will generate code in this style, however. |
| suggestion | When this style rule is violated, show it to the user as a suggestion. Suggestions appear as three grey dots under the first two characters. |
| warning | When this style rule is violated, show a compiler warning. |
| error | When this style rule is violated, show a compiler error. |

The following list shows the allowable language convention rules:

- .NET Code Style Settings
 - "This." and "Me." qualifiers
 - dotnet_style_qualification_for_field
 - dotnet_style_qualification_for_property
 - dotnet_style_qualification_for_method
 - dotnet_style_qualification_for_event
 - Language keywords instead of framework type names for type references
 - dotnet_style_predefined_type_for_locals_parameters_members
 - dotnet_style_predefined_type_for_member_access
 - Modifier preferences
 - dotnet_style_require_accessibility_modifiers
 - csharp_preferred_modifier_order
 - visual_basic_preferred_modifier_order
 - Expression-level preferences
 - dotnet_style_object_initializer
 - dotnet_style_collection_initializer
 - dotnet_style_explicit_tuple_names
 - dotnet_prefer_inferred_tuple_names
 - dotnet_prefer_inferred_anonymous_type_member_names
 - "Null" checking preferences
 - dotnet_style_coalesce_expression
 - dotnet_style_null_propagation
- C# Code Style Settings
 - Implicit and explicit types
 - csharp_style_var_for_builtin_types
 - csharp_style_var_when_type_is_apparent
 - csharp_style_var_elsewhere
 - Expression-bodied members
 - csharp_style_expression_bodied_methods
 - csharp_style_expression_bodied_constructors
 - csharp_style_expression_bodied_operators
 - csharp_style_expression_bodied_properties
 - csharp_style_expression_bodied_indexers
 - csharp_style_expression_bodied_accessors
 - Pattern matching
 - csharp_style_pattern_matching_over_is_with_cast_check
 - csharp_style_pattern_matching_over_as_with_null_check
 - Inlined variable declarations
 - csharp_style_inlined_variable_declaration
 - Expression-level preferences
 - csharp_prefer_simple_default_expression
 - csharp_style_deconstructed_variable_declaration
 - csharp_style_pattern_local_over_anonymous_function
 - "Null" checking preferences
 - csharp_style_throw_expression
 - csharp_style_conditional_delegate_call

- o [Code block preferences](#)
- o [csharp_prefer_braces](#)

.NET code style settings

The style rules in this section are applicable to both C# and Visual Basic. To see code examples in your preferred programming language, choose it in the drop-down **Language** menu at the top-right corner of your browser window.

"This." and "Me." qualifiers

This style rule (rule IDs IDE0003 and IDE0009) can be applied to fields, properties, methods or events. A value of **true** means prefer the code symbol to be prefaced with `this.` in C# or `Me.` in Visual Basic. A value of **false** means prefer the code element *not* to be prefaced with `this.` or `Me.`.

The following table shows the rule names, applicable programming languages, and default values:

| RULE NAME | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT VALUE |
|---|----------------------|-----------------------------|
| dotnet_style_qualification_for_field | C# and Visual Basic | false:none |
| dotnet_style_qualification_for_property | C# and Visual Basic | false:none |
| dotnet_style_qualification_for_method | C# and Visual Basic | false:none |
| dotnet_style_qualification_for_event | C# and Visual Basic | false:none |

dotnet_style_qualification_for_field

- When this rule is set to **true**, prefer fields to be prefaced with `this.` in C# or `Me.` in Visual Basic.
- When this rule is set to **false**, prefer fields *not* to be prefaced with `this.` or `Me.`.

Code examples:

```
// dotnet_style_qualification_for_field = true
this.capacity = 0;

// dotnet_style_qualification_for_field = false
capacity = 0;
```

```
' dotnet_style_qualification_for_field = true
Me.capacity = 0

' dotnet_style_qualification_for_field = false
capacity = 0
```

dotnet_style_qualification_for_property

- When this rule is set to **true**, prefer properties to be prefaced with `this.` in C# or `Me.` in Visual Basic.
- When this rule is set to **false**, prefer properties *not* to be prefaced with `this.` or `Me.`.

Code examples:

```
// dotnet_style_qualification_for_property = true
this.ID = 0;

// dotnet_style_qualification_for_property = false
ID = 0;
```

```
' dotnet_style_qualification_for_property = true
Me.ID = 0

' dotnet_style_qualification_for_property = false
ID = 0
```

dotnet_style_qualification_for_method

- When this rule is set to **true**, prefer methods to be prefaced with `this.` in C# or `Me.` in Visual Basic.
- When this rule is set to **false**, prefer methods *not* to be prefaced with `this.` or `Me.`.

Code examples:

```
// dotnet_style_qualification_for_method = true
this.Display();

// dotnet_style_qualification_for_method = false
Display();
```

```
' dotnet_style_qualification_for_method = true
Me.Display()

' dotnet_style_qualification_for_method = false
Display()
```

dotnet_style_qualification_for_event

- When this rule is set to **true**, prefer events to be prefaced with `this.` in C# or `Me.` in Visual Basic.
- When this rule is set to **false**, prefer events *not* to be prefaced with `this.` or `Me.`.

Code examples:

```
// dotnet_style_qualification_for_event = true
this.Elapsed += Handler;

// dotnet_style_qualification_for_event = false
Elapsed += Handler;
```

```
' dotnet_style_qualification_for_event = true
AddHandler Me.Elapsed, AddressOf Handler

' dotnet_style_qualification_for_event = false
AddHandler Elapsed, AddressOf Handler
```

These rules could appear in an `.editorconfig` file as follows:

```

# CSharp and Visual Basic code style settings:
[*.{cs,vb}]
dotnet_style_qualification_for_field = false:suggestion
dotnet_style_qualification_for_property = false:suggestion
dotnet_style_qualification_for_method = false:suggestion
dotnet_style_qualification_for_event = false:suggestion

```

Language keywords instead of framework type names for type references

This style rule can be applied to local variables, method parameters, and class members, or as a separate rule to type member access expressions. A value of **true** means prefer the language keyword (e.g. `int` or `Integer`) instead of the type name (e.g. `Int32`) for types that have a keyword to represent them. A value of **false** means prefer the type name instead of the language keyword.

The following table shows the rule names, rules IDs, applicable programming languages, and default values:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT |
|--|---------------------|----------------------|-----------------------|
| dotnet_style_predefined_type_for_locals_parameters_members | IDE0012 and IDE0014 | C# and Visual Basic | true:none |
| dotnet_style_predefined_type_for_member_access | IDE0013 and IDE0015 | C# and Visual Basic | true:none |

dotnet_style_predefined_type_for_locals_parameters_members

- When this rule is set to **true**, prefer the language keyword for local variables, method parameters, and class members, instead of the type name, for types that have a keyword to represent them.
- When this rule is set to **false**, prefer the type name for local variables, method parameters, and class members, instead of the language keyword.

Code examples:

```

// dotnet_style_predefined_type_for_locals_parameters_members = true
private int _member;

// dotnet_style_predefined_type_for_locals_parameters_members = false
private Int32 _member;

```

```

' dotnet_style_predefined_type_for_locals_parameters_members = true
Private _member As Integer

' dotnet_style_predefined_type_for_locals_parameters_members = false
Private _member As Int32

```

dotnet_style_predefined_type_for_member_access

- When this rule is set to **true**, prefer the language keyword for member access expressions, instead of the type name, for types that have a keyword to represent them.
- When this rule is set to **false**, prefer the type name for member access expressions, instead of the language keyword.

Code examples:

```
// dotnet_style_predefined_type_for_member_access = true
var local = int.MaxValue;

// dotnet_style_predefined_type_for_member_access = false
var local = Int32.MaxValue;
```

```
' dotnet_style_predefined_type_for_member_access = true
Dim local = Integer.MaxValue

' dotnet_style_predefined_type_for_member_access = false
Dim local = Int32.MaxValue
```

These rules could appear in an .editorconfig file as follows:

```
# CSharp and Visual Basic code style settings:
[*.{cs,vb}]
dotnet_style_predefined_type_for_locals_parameters_members = true:suggestion
dotnet_style_predefined_type_for_member_access = true:suggestion
```

Modifier preferences

The style rules in this section concern modifier preferences, including requiring accessibility modifiers and specifying the desired modifier sort order.

The following table shows the rule names, rule IDs, applicable programming languages, default values, and first supported version of Visual Studio:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|--|---------|----------------------|---|----------------------------|
| dotnet_style_require_accessibility_modifiers | IDE0040 | C# and Visual Basic | for_non_interface_members:none | 15.5 |
| csharp_preferred_modifier_order | IDE0036 | C# | public, private, protected, internal, static, extern, new, virtual, abstract, sealed, override, readonly, unsafe, volatile, async:none | 15.5 |
| visual_basic_preferred_modifier_order | IDE0036 | Visual Basic | Partial, Default, Private, Protected, Public, Friend, NotOverridable, Overridable, MustOverride, Overloads, Overrides, MustInherit, NotInheritable, Static, Shared, Shadows, ReadOnly, WriteOnly, Dim, Const, WithEvents, Widening, Narrowing, Custom, Async:none | 15.5 |

dotnet_style_require_accessibility_modifiers

This rule does not accept a **true** or **false** value; instead it accepts a value from the following table:

| VALUE | DESCRIPTION |
|---------------------------|--|
| always | Prefer accessibility modifiers to be specified |
| for_non_interface_members | Prefer accessibility modifiers to be declared except for public interface members. This will currently not differ from always and will act as future proofing for if C# adds default interface methods. |
| never | Do not prefer accessibility modifiers to be specified |

Code examples:

```
// dotnet_style_require_accessibility_modifiers = always
// dotnet_style_require_accessibility_modifiers = for_non_interface_members
class MyClass
{
    private const string thisFieldIsConst= "constant";
}

// dotnet_style_require_accessibility_modifiers = never
class MyClass
{
    const string thisFieldIsConst= "constant";
}
```

csharp_preferred_modifier_order

- When this rule is set to a list of modifiers, prefer the specified ordering.
- When this rule is omitted from the file, do not prefer a modifier order.

Code examples:

```
// csharp_preferred_modifier_order =
public,private,protected,internal,static,extern,new,virtual,abstract,sealed,override,readonly,unsafe,volatile,
async
class MyClass
{
    private static readonly int _daysInYear = 365;
}
```

visual_basic_preferred_modifier_order

- When this rule is set to a list of modifiers, prefer the specified ordering.
- When this rule is omitted from the file, do not prefer a modifier order.

Code examples:

```
' visual_basic_preferred_modifier_order =
Partial,Default,Private,Protected,Public,Friend,NotOverridable,Overridable,MustOverride,Overloads,Overrides,Mu
stInherit,NotInheritable,Static,Shared,Shadows,ReadOnly,WriteOnly,Dim,Const,WithEvents,Widening,Narrowing,Cust
om,Async
Public Class MyClass
    Private Shared ReadOnly daysInYear As Int = 365
End Class
```

These rules could appear in an .editorconfig file as follows:

```

# CSharp and Visual Basic code style settings:
[*.{cs,vb}]
dotnet_style_require_accessibility_modifiers = always:suggestion

# CSharp code style settings:
[*.cs]
csharp_preferred_modifier_order =
public,private,protected,internal,static,extern,new,virtual,abstract,sealed,override,readonly,unsafe,volatile,
async:suggestion

# Visual Basic code style settings:
[*.vb]
visual_basic_preferred_modifier_order =
Partial,Default,Private,Protected,Public,Friend,NotOverridable,Overridable,MustOverride,Overloads,Overrides,Mu
stInherit,NotInheritable,Static,Shared,Shadows,ReadOnly,WriteOnly,Dim,Const,WithEvents,Widening,Narrowing,Cust
om,Async:suggestion

```

Expression-level preferences

The style rules in this section concern expression-level preferences, including the use of object initializers, collection initializers, explicit or inferred tuple names, and inferred anonymous types.

The following table shows the rule names, rule IDs, applicable programming languages, default values, and first supported version of Visual Studio:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|--|---------|------------------------------|-----------------------|----------------------------|
| dotnet_style_object_initializer | IDE0017 | C# and Visual Basic | true:suggestion | First release |
| dotnet_style_collection_initializer | IDE0028 | C# and Visual Basic | true:suggestion | First release |
| dotnet_style_explicit_tuple_names | IDE0033 | C# 7.0+ and Visual Basic 15+ | true:suggestion | First release |
| dotnet_style_prefer_inferred_tuple_names | IDE0037 | C# 7.1+ and Visual Basic 15+ | true:suggestion | 15.6 |
| dotnet_style_prefer_inferred_anonymous_type_member_names | IDE0037 | C# and Visual Basic | true:suggestion | 15.6 |

dotnet_style_object_initializer

- When this rule is set to **true**, prefer objects to be initialized using object initializers when possible.
- When this rule is set to **false**, prefer objects to *not* be initialized using object initializers.

Code examples:

```

// dotnet_style_object_initializer = true
var c = new Customer() { Age = 21 };

// dotnet_style_object_initializer = false
var c = new Customer();
c.Age = 21;

```

```
' dotnet_style_object_initializer = true
Dim c = New Customer() With {.Age = 21}

' dotnet_style_object_initializer = false
Dim c = New Customer()
c.Age = 21
```

dotnet_style_collection_initializer

- When this rule is set to **true**, prefer collections to be initialized using collection initializers when possible.
- When this rule is set to **false**, prefer collections to *not* be initialized using collection initializers.

Code examples:

```
// dotnet_style_collection_initializer = true
var list = new List<int> { 1, 2, 3 };

// dotnet_style_collection_initializer = false
var list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);
```

```
' dotnet_style_collection_initializer = true
Dim list = New List(Of Integer) From {1, 2, 3}

' dotnet_style_collection_initializer = false
Dim list = New List(Of Integer)
list.Add(1)
list.Add(2)
list.Add(3)
```

dotnet_style_explicit_tuple_names

- When this rule is set to **true**, prefer tuple names to ItemX properties.
- When this rule is set to **false**, prefer ItemX properties to tuple names.

Code examples:

```
// dotnet_style_explicit_tuple_names = true
(string name, int age) customer = GetCustomer();
var name = customer.name;

// dotnet_style_explicit_tuple_names = false
(string name, int age) customer = GetCustomer();
var name = customer.Item1;
```

```
' dotnet_style_explicit_tuple_names = true
Dim customer As (name As String, age As Integer) = GetCustomer()
Dim name = customer.name

' dotnet_style_explicit_tuple_names = false
Dim customer As (name As String, age As Integer) = GetCustomer()
Dim name = customer.Item1
```

dotnet_style_prefer_inferred_tuple_names

- When this rule is set to **true**, prefer inferred tuple element names.

- When this rule is set to **false**, prefer explicit tuple element names.

Code examples:

```
// dotnet_style_prefer_inferred_tuple_names = true
var tuple = (age, name);

// dotnet_style_prefer_inferred_tuple_names = false
var tuple = (age: age, name: name);
```

dotnet_style_prefer_inferred_anonymous_type_member_names

- When this rule is set to **true**, prefer inferred anonymous type member names.
- When this rule is set to **false**, prefer explicit anonymous type member names.

Code examples:

```
// dotnet_style_prefer_inferred_anonymous_type_member_names = true
var anon = new { age, name };

// dotnet_style_prefer_inferred_anonymous_type_member_names = false
var anon = new { age = age, name = name };
```

These rules could appear in an .editorconfig file as follows:

```
# CSharp and Visual Basic code style settings:
[*.{cs,vb}]
dotnet_style_object_initializer = true:suggestion
dotnet_style_collection_initializer = true:suggestion
dotnet_style_explicit_tuple_names = true:suggestion
dotnet_style_prefer_inferred_tuple_names = true:suggestion
dotnet_style_prefer_inferred_anonymous_type_member_names = true:suggestion
```

Null-checking preferences

The style rules in this section concern null-checking preferences.

The following table shows the rule names, rule IDs, applicable programming languages, default values, and first supported version of Visual Studio:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|----------------------------------|---------|------------------------------|-----------------------|----------------------------|
| dotnet_style_coalesce_expression | IDE0029 | C# and Visual Basic | true:suggestion | First release |
| dotnet_style_null_propagation | IDE0031 | C# 6.0+ and Visual Basic 14+ | true:suggestion | First release |

dotnet_style_coalesce_expression

- When this rule is set to **true**, prefer null coalescing expressions to ternary operator checking.
- When this rule is set to **false**, prefer ternary operator checking to null coalescing expressions.

Code examples:

```
// dotnet_style_coalesce_expression = true
var v = x ?? y;

// dotnet_style_coalesce_expression = false
var v = x != null ? x : y; // or
var v = x == null ? y : x;
```

```
' dotnet_style_coalesce_expression = true
Dim v = If(x, y)

' dotnet_style_coalesce_expression = false
Dim v = If(x Is Nothing, y, x) ' or
Dim v = If(x IsNot Nothing, y, x)
```

dotnet_style_null_propagation

- When this rule is set to **true**, prefer to use null-conditional operator when possible.
- When this rule is set to **false**, prefer to use ternary null checking where possible.

Code examples:

```
// dotnet_style_null_propagation = true
var v = o?.ToString();

// dotnet_style_null_propagation = false
var v = o == null ? null : o.ToString(); // or
var v = o != null ? o.ToString() : null;
```

```
' dotnet_style_null_propagation = true
Dim v = o?.ToString()

' dotnet_style_null_propagation = false
Dim v = If(o Is Nothing, Nothing, o.ToString()) ' or
Dim v = If(o IsNot Nothing, o.ToString(), Nothing)
```

These rules could appear in an .editorconfig file as follows:

```
# CSharp and Visual Basic code style settings:
[*.{cs,vb}]
dotnet_style_coalesce_expression = true:suggestion
dotnet_style_null_propagation = true:suggestion
```

C# code style settings

The style rules in this section are applicable to C# only.

Implicit and explicit types

The style rules in this section (rule IDs IDE0007 and IDE0008) concern the use of the `var` keyword versus an explicit type in a variable declaration. This rule can be applied separately to built-in types, when the type is apparent, and elsewhere.

The following table shows the rule names, applicable programming languages, and default values:

| RULE NAME | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT |
|------------------------------------|----------------------|-----------------------|
| csharp_style_var_for_builtin_types | C# | true:none |

| RULE NAME | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT |
|--|----------------------|-----------------------|
| csharp_style_var_when_type_is_apparent | C# | true:none |
| csharp_style_var_elsewhere | C# | true:none |

csharp_style_var_for_built_in_types

- When this rule is set to **true**, prefer `var` is used to declare variables with built-in system types such as `int`.
- When this rule is set to **false**, prefer explicit type over `var` to declare variables with built-in system types such as `int`.

Code examples:

```
// csharp_style_var_for_built_in_types = true
var x = 5;

// csharp_style_var_for_built_in_types = false
int x = 5;
```

csharp_style_var_when_type_is_apparent

- When this rule is set to **true**, prefer `var` when the type is already mentioned on the right-hand side of a declaration expression.
- When this rule is set to **false**, prefer explicit type over `var` when the type is already mentioned on the right-hand side of a declaration expression.

Code examples:

```
// csharp_style_var_when_type_is_apparent = true
var obj = new Customer();

// csharp_style_var_when_type_is_apparent = false
Customer obj = new Customer();
```

csharp_style_var_elsewhere

- When this rule is set to **true**, prefer `var` over explicit type in all cases, unless overridden by another code style rule.
- When this rule is set to **false**, prefer explicit type over `var` in all cases, unless overridden by another code style rule.

Code examples:

```
// csharp_style_var_elsewhere = true
var f = this.Init();

// csharp_style_var_elsewhere = false
bool f = this.Init();
```

Example .editorconfig file:

```

# CSharp code style settings:
[*.cs]
csharp_style_var_for_builtin_types = true:suggestion
csharp_style_var_when_type_is_apparent = true:suggestion
csharp_style_var_elsewhere = true:suggestion

```

Expression-bodied members

The style rules in this section concern the use of [expression-bodied members](#) when the logic consists of a single expression. This rule can be applied to methods, constructors, operators, properties, indexers, and accessors.

The following table shows the rule names, rule IDs, applicable language versions, default values, and first supported version of Visual Studio:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|---|---------------------|----------------------|-----------------------|----------------------------|
| csharp_style_expression_bodied_methods | IDE0022 | C# 6.0+ | false:none | 15.3 |
| csharp_style_expression_bodied_constructors | IDE0021 | C# 7.0+ | false:none | 15.3 |
| csharp_style_expression_bodied_operators | IDE0023 and IDE0024 | C# 7.0+ | false:none | 15.3 |
| csharp_style_expression_bodied_properties | IDE0025 | C# 7.0+ | true:none | 15.3 |
| csharp_style_expression_bodied_indexers | IDE0026 | C# 7.0+ | true:none | 15.3 |
| csharp_style_expression_bodied_accessors | IDE0027 | C# 7.0+ | true:none | 15.3 |

csharp_style_expression_bodied_methods

This rule accepts values from the following table:

| VALUE | DESCRIPTION |
|---------------------|--|
| true | Prefer expression-bodied members for methods |
| when_on_single_line | Prefer expression-bodied members for methods when they will be a single line |
| false | Prefer block bodies for methods |

Code examples:

```

// csharp_style_expression_bodied_methods = true
public int GetAge() => this.Age;

// csharp_style_expression_bodied_methods = false
public int GetAge() { return this.Age; }

```

csharp_style_expression_bodied_constructors

This rule accepts values from the following table:

| VALUE | DESCRIPTION |
|---------------------|---|
| true | Prefer expression-bodied members for constructors |
| when_on_single_line | Prefer expression-bodied members for constructors when they will be a single line |
| false | Prefer block bodies for constructors |

Code examples:

```
// csharp_style_expression_bodied_constructors = true
public Customer(int age) => Age = age;

// csharp_style_expression_bodied_constructors = false
public Customer(int age) { Age = age; }
```

csharp_style_expression_bodied_operators

This rule accepts values from the following table:

| VALUE | DESCRIPTION |
|---------------------|--|
| true | Prefer expression-bodied members for operators |
| when_on_single_line | Prefer expression-bodied members for operators when they will be a single line |
| false | Prefer block bodies for operators |

Code examples:

```
// csharp_style_expression_bodied_operators = true
public static ComplexNumber operator + (ComplexNumber c1, ComplexNumber c2)
    => new ComplexNumber(c1.Real + c2.Real, c1.Imaginary + c2.Imaginary);

// csharp_style_expression_bodied_operators = false
public static ComplexNumber operator + (ComplexNumber c1, ComplexNumber c2)
{ return new ComplexNumber(c1.Real + c2.Real, c1.Imaginary + c2.Imaginary); }
```

csharp_style_expression_bodied_properties

This rule accepts values from the following table:

| VALUE | DESCRIPTION |
|---------------------|---|
| true | Prefer expression-bodied members for properties |
| when_on_single_line | Prefer expression-bodied members for properties when they will be a single line |
| false | Prefer block bodies for properties |

Code examples:

```
// csharp_style_expression_bodied_properties = true
public int Age => _age;

// csharp_style_expression_bodied_properties = false
public int Age { get { return _age; } }
```

csharp_style_expression_bodied_indexers

This rule accepts values from the following table:

| VALUE | DESCRIPTION |
|---------------------|---|
| true | Prefer expression-bodied members for indexers |
| when_on_single_line | Prefer expression-bodied members for indexers when they will be a single line |
| false | Prefer block bodies for indexers |

Code examples:

```
// csharp_style_expression_bodied_indexers = true
public T this[int i] => _value[i];

// csharp_style_expression_bodied_indexers = false
public T this[int i] { get { return _values[i]; } }
```

csharp_style_expression_bodied_accessors

This rule accepts values from the following table:

| VALUE | DESCRIPTION |
|---------------------|--|
| true | Prefer expression-bodied members for accessors |
| when_on_single_line | Prefer expression-bodied members for accessors when they will be a single line |
| false | Prefer block bodies for accessors |

Code examples:

```
// csharp_style_expression_bodied_accessors = true
public int Age { get => _age; set => _age = value; }

// csharp_style_expression_bodied_accessors = false
public int Age { get { return _age; } set { _age = value; } }
```

Example .editorconfig file:

```

# CSharp code style settings:
[*.cs]
csharp_style_expression_bodied_methods = false:none
csharp_style_expression_bodied_constructors = false:none
csharp_style_expression_bodied_operators = false:none
csharp_style_expression_bodied_properties = true:suggestion
csharp_style_expression_bodied_indexers = true:suggestion
csharp_style_expression_bodied_accessors = true:suggestion

```

Pattern matching

The style rules in this section concern the use of [pattern matching](#) in C#.

The following table shows the rule names, rule IDs, applicable language versions, and default values:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT |
|---|---------|----------------------|-----------------------|
| csharp_style_pattern_matching_over_is_with_cast_check | IDE0020 | C# 7.0+ | true:suggestion |
| csharp_style_pattern_matching_over_as_with_null_check | IDE0019 | C# 7.0+ | true:suggestion |

csharp_style_pattern_matching_over_is_with_cast_check

- When this rule is set to **true**, prefer pattern matching instead of `is` expressions with type casts.
- When this rule is set to **false**, prefer `is` expressions with type casts instead of pattern matching.

Code examples:

```

// csharp_style_pattern_matching_over_is_with_cast_check = true
if (o is int i) {...}

// csharp_style_pattern_matching_over_is_with_cast_check = false
if (o is int) {var i = (int)o; ... }

```

csharp_style_pattern_matching_over_as_with_null_check

- When this rule is set to **true**, prefer pattern matching instead of `as` expressions with null checks to determine if something is of a particular type.
- When this rule is set to **false**, prefer `as` expressions with null checks instead of pattern matching to determine if something is of a particular type.

Code examples:

```

// csharp_style_pattern_matching_over_as_with_null_check = true
if (o is string s) {...}

// csharp_style_pattern_matching_over_as_with_null_check = false
var s = o as string;
if (s != null) {...}

```

Example .editorconfig file:

```
# CSharp code style settings:
[*.cs]
csharp_style_pattern_matching_over_is_with_cast_check = true:suggestion
csharp_style_pattern_matching_over_as_with_null_check = true:suggestion
```

Inlined variable declarations

This style rule concerns whether `out` variables are declared inline or not. Starting in C# 7, you can [declare an out variable in the argument list of a method call](#), rather than in a separate variable declaration.

The following table shows the rule name, rule ID, applicable language versions, and default values:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT |
|---|---------|----------------------|-----------------------|
| csharp_style_inlined_variable_declaration | IDE0018 | C# 7.0+ | true:suggestion |

csharp_style_inlined_variable_declaration

- When this rule is set to **true**, prefer `out` variables to be declared inline in the argument list of a method call when possible.
- When this rule is set to **false**, prefer `out` variables to be declared before the method call.

Code examples:

```
// csharp_style_inlined_variable_declaration = true
if (int.TryParse(value, out int i) {...}

// csharp_style_inlined_variable_declaration = false
int i;
if (int.TryParse(value, out i) {...}
```

Example .editorconfig file:

```
# CSharp code style settings:
[*.cs]
csharp_style_inlined_variable_declaration = true:suggestion
```

Expression-level preferences

The style rules in this section concern expression-level preferences, including the use of [default expressions](#), deconstructed variables, and local functions over anonymous functions.

The following table shows the rule name, rule ID, applicable language versions, default values, and first supported version of Visual Studio:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|---|---------|----------------------|-----------------------|----------------------------|
| csharp_prefer_simple_default_expression | IDE0034 | C# 7.1+ | true:suggestion | 15.3 |
| csharp_style_deconstructed_variable_declaration | IDE0042 | C# 7.0+ | true:suggestion | 15.5 |

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|--|---------|----------------------|-----------------------|----------------------------|
| csharp_style_pattern_local_over_anonymous_function | IDE0039 | C# 7.0+ | true:suggestion | 15.5 |

csharp_prefer_simple_default_expression

This style rule concerns using the `default` literal for default value expressions when the compiler can infer the type of the expression.

- When this rule is set to **true**, prefer `default` over `default(T)`.
- When this rule is set to **false**, prefer `default(T)` over `default`.

Code examples:

```
// csharp_prefer_simple_default_expression = true
void DoWork(CancellationToken cancellationToken = default) { ... }

// csharp_prefer_simple_default_expression = false
void DoWork(CancellationToken cancellationToken = default(CancellationToken)) { ... }
```

csharp_style_deconstructed_variable_declaration

- When this rule is set to **true**, prefer deconstructed variable declaration.
- When this rule is set to **false**, do not prefer deconstruction in variable declarations.

Code examples:

```
// csharp_style_deconstructed_variable_declaration = true
var (name, age) = GetPersonTuple();
Console.WriteLine($"{name} {age}");

(int x, int y) = GetPointTuple();
Console.WriteLine($"{x} {y}");

// csharp_style_deconstructed_variable_declaration = false
var person = GetPersonTuple();
Console.WriteLine($"{person.name} {person.age}");

(int x, int y) point = GetPointTuple();
Console.WriteLine($"{point.x} {point.y}");
```

csharp_style_pattern_local_over_anonymous_function

- When this rule is set to **true**, prefer local functions over anonymous functions.
- When this rule is set to **false**, prefer anonymous functions over local functions.

Code examples:

```
// csharp_style_pattern_local_over_anonymous_function = true
int fibonacci(int n)
{
    return n <= 1 ? 1 : fibonacci(n-1) + fibonacci(n-2);
}

// csharp_style_pattern_local_over_anonymous_function = false
Func<int, int> fibonacci = null;
fibonacci = (int n) =>
{
    return n <= 1 ? 1 : fibonacci(n - 1) + fibonacci(n - 2);
};
```

Example .editorconfig file:

```
# CSharp code style settings:
[*.cs]
csharp_prefer_simple_default_expression = true:suggestion
csharp_style_deconstructed_variable_declaration = true:suggestion
csharp_style_pattern_local_over_anonymous_function = true:suggestion
```

"Null" checking preferences

These style rules concern the syntax around `null` checking, including using `throw` expressions or `throw` statements, and whether to perform a null check or use the conditional coalescing operator (`?.`) when invoking a [lambda expression](#).

The following table shows the rule names, rule IDs, applicable language versions, and default values:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT |
|--|---------|----------------------|-----------------------|
| csharp_style_throw_expression | IDE0016 | C# 7.0+ | true:suggestion |
| csharp_style_conditional_delegate_call | IDE0041 | C# 6.0+ | true:suggestion |

csharp_style_throw_expression

- When this rule is set to **true**, prefer to use `throw` expressions instead of `throw` statements.
- When this rule is set to **false**, prefer to use `throw` statements instead of `throw` expressions.

Code examples:

```
// csharp_style_throw_expression = true
this.s = s ?? throw new ArgumentNullException(nameof(s));

// csharp_style_throw_expression = false
if (s == null) { throw new ArgumentNullException(nameof(s)); }
this.s = s;
```

csharp_style_conditional_delegate_call

- When this rule is set to **true**, prefer to use the conditional coalescing operator (`?.`) when invoking a lambda expression, instead of performing a null check.
- When this rule is set to **false**, prefer to perform a null check before invoking a lambda expression, instead of using the conditional coalescing operator (`?.`).

Code examples:

```
// csharp_style_conditional_delegate_call = true  
func?.Invoke(args);  
  
// csharp_style_conditional_delegate_call = false  
if (func != null) { func(args); }
```

Example .editorconfig file:

```
# CSharp code style settings:  
[*.cs]  
csharp_style_throw_expression = true:suggestion  
csharp_style_conditional_delegate_call = false:suggestion
```

Code block preferences

This style rule concerns the use of curly braces `{ }` to surround code blocks.

The following table shows the rule name, rule ID, applicable language versions, default values, and first supported version of Visual Studio:

| RULE NAME | RULE ID | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|----------------------|---------|----------------------|-----------------------|----------------------------|
| csharp_prefer_braces | IDE0011 | C# | true:none | 15.3 |

csharp_prefer_braces

- When this rule is set to **true**, prefer curly braces even for one line of code.
- When this rule is set to **false**, prefer no curly braces if allowed.

Code examples:

```
// csharp_prefer_braces = true  
if (test) { this.Display(); }  
  
// csharp_prefer_braces = false  
if (test) this.Display();
```

Example .editorconfig file:

```
# CSharp code style settings:  
[*.cs]  
csharp_prefer_braces = true:none
```

Formatting conventions

Most of the rules for formatting conventions have the following format:

```
rule_name = false|true
```

You specify either **true** (prefer this style) or **false** (do not prefer this style). You do not specify a severity. For a few rules, instead of true or false, you specify other values to describe when and where to apply the rule.

The following list shows the formatting convention rules available in Visual Studio:

- .NET Formatting Settings

- [Organize Usings](#)
 - dotnet_sort_system_directives_first
- C# Formatting Settings
 - [Newline Options](#)
 - csharp_new_line_before_open_brace
 - csharp_new_line_before_else
 - csharp_new_line_before_catch
 - csharp_new_line_before_finally
 - csharp_new_line_before_members_in_object_initializers
 - csharp_new_line_before_members_in_anonymous_types
 - csharp_new_line_between_query_expression_clauses
 - [Indentation Options](#)
 - csharp_indent_case_contents
 - csharp_indent_switch_labels
 - csharp_indent_labels
 - [Spacing Options](#)
 - csharp_space_after_cast
 - csharp_space_after_keywords_in_control_flow_statements
 - csharp_space_between_method_declaration_parameter_list_parentheses
 - csharp_space_between_method_call_parameter_list_parentheses
 - csharp_space_between_parentheses
 - [Wrapping Options](#)
 - csharp_preserve_single_line_statements
 - csharp_preserve_single_line_blocks

.NET formatting settings

The formatting rules in this section are applicable to C# and Visual Basic.

Organize usings

This formatting rule concerns the placement of System.* using directives with respect to other using directives.

The following table shows the rule name, applicable languages, default value, and first supported version of Visual Studio:

| RULE NAME | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|-------------------------------------|----------------------|-----------------------|----------------------------|
| dotnet_sort_system_directives_first | C# and Visual Basic | true | 15.3 |

dotnet_sort_system_directives_first

- When this rule is set to **true**, sort System.* using directives alphabetically, and place them before other usings.
- When this rule is set to **false**, do not place System.* using directives before other using directives.

Code examples:

```
// dotnet_sort_system_directives_first = true
using System.Collections.Generic;
using System.Threading.Tasks;
using Octokit;

// dotnet_sort_system_directives_first = false
using System.Collections.Generic;
using Octokit;
using System.Threading.Tasks;
```

Example .editorconfig file:

```
# .NET formatting settings:
[*.{cs,vb}]
dotnet_sort_system_directives_first = true
```

C# formatting settings

The formatting rules in this section apply only to C# code.

Newline Options

These formatting rules concern the use of new lines to format code.

The following table shows the "new line" rule names, applicable languages, default values, and first supported version of Visual Studio:

| RULE NAME | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|---|----------------------|-----------------------|----------------------------|
| csharp_new_line_before_open_brace | C# | all | 15.3 |
| csharp_new_line_before_else | C# | true | 15.3 |
| csharp_new_line_before_catch | C# | true | 15.3 |
| csharp_new_line_before_finally | C# | true | 15.3 |
| csharp_new_line_before_members_in_object_initializers | C# | true | 15.3 |
| csharp_new_line_before_members_in_anonymous_types | C# | true | 15.3 |
| csharp_new_line_between_query_expression_clauses | C# | true | 15.3 |

csharp_new_line_before_open_brace

This rule concerns whether an open brace `{` should be placed on the same line as the preceding code, or on a new line. For this rule, you do not specify **true** or **false**. Instead you specify **all**, **none**, or one or more code elements such as **methods** or **properties**, to define when this rule should be applied. The complete list of allowable values is shown in the following table:

| Value | Description |
|---|---|
| accessors, anonymous_methods, anonymous_types, control_blocks, events, indexers, lambdas, local_functions, methods, object_collection, properties, types. (For multiple kinds, separate with ','). | Require braces to be on a new line for the specified code elements (also known as "Allman" style) |
| all | Require braces to be on a new line for all expressions ("Allman" style) |
| none | Require braces to be on the same line for all expressions ("K&R") |

Code examples:

```
// csharp_new_line_before_open_brace = all
void MyMethod()
{
    if (...) {
        ...
    }
}

// csharp_new_line_before_open_brace = none
void MyMethod() {
    if (...) {
        ...
    }
}
```

csharp_new_line_before_else

- When this rule is set to **true**, place `else` statements on a new line.
- When this rule is set to **false**, place `else` statements on the same line.

Code examples:

```
// csharp_new_line_before_else = true
if (...) {
    ...
}
else {
    ...
}

// csharp_new_line_before_else = false
if (...) {
    ...
} else {
    ...
}
```

csharp_new_line_before_catch

- When this rule is set to **true**, place `catch` statements on a new line.
- When this rule is set to **false**, place `catch` statements on the same line.

Code examples:

```
// csharp_new_line_before_catch = true
try {
    ...
}
catch (Exception e) {
    ...
}

// csharp_new_line_before_catch = false
try {
    ...
} catch (Exception e) {
    ...
}
```

csharp_new_line_before_finally

- When this rule is set to **true**, require `finally` statements to be on a new line after the closing brace.
- When this rule is set to **false**, require `finally` statements to be on the same line as the closing brace.

Code examples:

```
// csharp_new_line_before_finally = true
try {
    ...
}
catch (Exception e) {
    ...
}
finally {
    ...
}

// csharp_new_line_before_finally = false
try {
    ...
} catch (Exception e) {
    ...
} finally {
    ...
}
```

csharp_new_line_before_members_in_object_initializers

- When this rule is set to **true**, require members of object initializers to be on separate lines.
- When this rule is set to **false**, require members of object initializers to be on the same line.

Code examples:

```
// csharp_new_line_before_members_in_object_initializers = true
var z = new B()
{
    A = 3,
    B = 4
}

// csharp_new_line_before_members_in_object_initializers = false
var z = new B()
{
    A = 3, B = 4
}
```

csharp_new_line_before_members_in_anonymous_types

- When this rule is set to **true**, require members of anonymous types to be on separate lines.
- When this rule is set to **false**, require members of anonymous types to be on the same line.

Code examples:

```
// csharp_new_line_before_members_in_anonymous_types = true
var z = new
{
    A = 3,
    B = 4
}

// csharp_new_line_before_members_in_anonymous_types = false
var z = new
{
    A = 3, B = 4
}
```

csharp_new_line_between_query_expression_clauses

- When this rule is set to **true**, require elements of query expression clauses to be on separate lines.
- When this rule is set to **false**, require elements of query expression clauses to be on the same line.

Code examples:

```
// csharp_new_line_between_query_expression_clauses = true
var q = from a in e
        from b in e
        select a * b;

// csharp_new_line_between_query_expression_clauses = false
var q = from a in e from b in e
        select a * b;
```

Example .editorconfig file:

```
# CSharp formatting settings:
[*.cs]
csharp_new_line_before_open_brace = methods, properties, control_blocks, types
csharp_new_line_before_else = true
csharp_new_line_before_catch = true
csharp_new_line_before_finally = true
csharp_new_line_before_members_in_object_initializers = true
csharp_new_line_before_members_in_anonymous_types = true
csharp_new_line_between_query_expression_clauses = true
```

Indentation options

These formatting rules concern the use of indentation to format code.

The following table shows the rule names, applicable languages, default values, and first supported version of Visual Studio:

| RULE NAME | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|-----------------------------|----------------------|-----------------------|----------------------------|
| csharp_indent_case_contents | C# | true | 15.3 |
| csharp_indent_switch_labels | C# | true | 15.3 |

| RULE NAME | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|----------------------|----------------------|-----------------------|----------------------------|
| csharp_indent_labels | C# | no_change | 15.3 |

csharp_indent_case_contents

- When this rule is set to **true**, indent `switch` case contents.
- When this rule is set to **false**, do not indent `switch` case contents.

Code examples:

```
// csharp_indent_case_contents = true
switch(c) {
    case Color.Red:
        Console.WriteLine("The color is red");
        break;
    case Color.Blue:
        Console.WriteLine("The color is blue");
        break;
    default:
        Console.WriteLine("The color is unknown.");
        break;
}

// csharp_indent_case_contents = false
switch(c) {
    case Color.Red:
        Console.WriteLine("The color is red");
        break;
    case Color.Blue:
        Console.WriteLine("The color is blue");
        break;
    default:
        Console.WriteLine("The color is unknown.");
        break;
}
```

csharp_indent_switch_labels

- When this rule is set to **true**, indent `switch` labels.
- When this rule is set to **false**, do not indent `switch` labels.

Code examples:

```
// csharp_indent_switch_labels = true
switch(c) {
    case Color.Red:
        Console.WriteLine("The color is red");
        break;
    case Color.Blue:
        Console.WriteLine("The color is blue");
        break;
    default:
        Console.WriteLine("The color is unknown.");
        break;
}

// csharp_indent_switch_labels = false
switch(c) {
case Color.Red:
    Console.WriteLine("The color is red");
    break;
case Color.Blue:
    Console.WriteLine("The color is blue");
    break;
default:
    Console.WriteLine("The color is unknown.");
    break;
}
```

csharp_indent_labels

This rule does not accept a **true** or **false** value; instead it accepts a value from the following table:

| VALUE | DESCRIPTION |
|-----------------------|---|
| flush_left | Labels are placed at the leftmost column |
| one_less_than_current | Labels are placed at one less indent to the current context |
| no_change | Labels are placed at the same indent as the current context |

Code examples:

```

// csharp_indent_labels= flush_left
class C
{
    private string MyMethod(...)
    {
        if (...) {
            goto error;
        }
    error:
        throw new Exception(...);
    }
}

// csharp_indent_labels = one_less_than_current
class C
{
    private string MyMethod(...)
    {
        if (...) {
            goto error;
        }
    error:
        throw new Exception(...);
    }
}

// csharp_indent_labels= no_change
class C
{
    private string MyMethod(...)
    {
        if (...) {
            goto error;
        }
    error:
        throw new Exception(...);
    }
}

```

Example .editorconfig file:

```

# CSharp formatting settings:
[*.cs]
csharp_indent_case_contents = true
csharp_indent_switch_labels = true
csharp_indent_labels = flush_left

```

Spacing Options

These formatting rules concern the use of space characters to format code.

The following table shows the rule names, applicable languages, default values, and first supported version of Visual Studio:

| Rule Name | Applicable Languages | Visual Studio Default | Visual Studio 2017 Version |
|--|----------------------|-----------------------|----------------------------|
| csharp_space_after_cast | C# | false | 15.3 |
| csharp_space_after_keyword s_in_control_flow_statement s | C# | true | 15.3 |

| RULE NAME | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|--|----------------------|-----------------------|----------------------------|
| csharp_space_between_meth od_declaration_parameter_lis t_parentheses | C# | false | 15.3 |
| csharp_space_between_meth od_call_parameter_list_paren theses | C# | false | 15.3 |
| csharp_space_between_pare ntheses | C# | false | 15.3 |

csharp_space_after_cast

- When this rule is set to **true**, require a space between a cast and the value.
- When this rule is set to **false**, require *no* space between the cast and the value.

Code examples:

```
// csharp_space_after_cast = true
int y = (int) x;

// csharp_space_after_cast = false
int y = (int)x;
```

csharp_space_after_keywords_in_control_flow_statements

- When this rule is set to **true**, require a space after a keyword in a control flow statement such as a `for` loop.
- When this rule is set to **false**, require *no* space after a keyword in a control flow statement such as a `for` loop.

Code examples:

```
// csharp_space_after_keywords_in_control_flow_statements = true
for (int i;i<x;i++) { ... }

// csharp_space_after_keywords_in_control_flow_statements = false
for(int i;i<x;i++) { ... }
```

csharp_space_between_method_declaration_parameter_list_parentheses

- When this rule is set to **true**, place a space character after the opening parenthesis and before the closing parenthesis of a method declaration parameter list.
- When this rule is set to **false**, do not place space characters after the opening parenthesis and before the closing parenthesis of a method declaration parameter list.

Code examples:

```
// csharp_space_between_method_declaration_parameter_list_parentheses = true
void Bark( int x ) { ... }

// csharp_space_between_method_declaration_parameter_list_parentheses = false
void Bark(int x) { ... }
```

csharp_space_between_method_call_parameter_list_parentheses

- When this rule is set to **true**, place a space character after the opening parenthesis and before the closing

parenthesis of a method call.

- When this rule is set to **false**, do not place space characters after the opening parenthesis and before the closing parenthesis of a method call.

Code examples:

```
// csharp_space_between_method_call_parameter_list_parentheses = true  
MyMethod( argument );  
  
// csharp_space_between_method_call_parameter_list_parentheses = false  
MyMethod(argument);
```

csharp_space_between_parentheses

This rule accepts one or more values from the following table:

| VALUE | DESCRIPTION |
|-------------------------|--|
| control_flow_statements | Place space between parentheses of control flow statements |
| expressions | Place space between parentheses of expressions |
| type_casts | Place space between parentheses in type casts |

If you omit this rule, or use a value other than `control_flow_statements`, `expressions`, or `type_casts`, the setting is not applied.

Code examples:

```
// csharp_space_between_parentheses = control_flow_statements  
for ( int i = 0; i < 10; i++ ) { }  
  
// csharp_space_between_parentheses = expressions  
var z = ( x * y ) - ( ( y - x ) * 3 );  
  
// csharp_space_between_parentheses = type_casts  
int y = ( int )x;
```

Example .editorconfig file:

```
# CSharp formatting settings:  
[*.cs]  
csharp_space_after_cast = true  
csharp_space_after_keywords_in_control_flow_statements = true  
csharp_space_between_method_declaration_parameter_list_parentheses = true  
csharp_space_between_method_call_parameter_list_parentheses = true  
csharp_space_between_parentheses = control_flow_statements, type_casts
```

Wrapping options

These formatting rules concern the use of single lines versus separate lines for statements and code blocks.

The following table shows the rule names, applicable languages, default values, and first supported version of Visual Studio:

| RULE NAME | APPLICABLE LANGUAGES | VISUAL STUDIO DEFAULT | VISUAL STUDIO 2017 VERSION |
|--|----------------------|-----------------------|----------------------------|
| csharp_preserve_single_line_statements | C# | true | 15.3 |
| csharp_preserve_single_line_blocks | C# | true | 15.3 |

csharp_preserve_single_line_statements

- When this rule is set to **true**, leave statements and member declarations on the same line.
- When this rule is set to **false**, leave statements and member declarations on different lines.

Code examples:

```
//csharp_preserve_single_line_statements = true
int i = 0; string name = "John";

//csharp_preserve_single_line_statements = false
int i = 0;
string name = "John";
```

csharp_preserve_single_line_blocks

- When this rule is set to **true**, leave code block on single line.
- When this rule is set to **false**, leave code block on separate lines.

Code examples:

```
//csharp_preserve_single_line_blocks = true
public int Foo { get; set; }

//csharp_preserve_single_line_blocks = false
public int MyProperty
{
    get; set;
}
```

Example .editorconfig file:

```
# CSharp formatting settings:
[*.cs]
csharp_preserve_single_line_statements = true
csharp_preserve_single_line_blocks = true
```

See also

- [Quick Actions](#)
- [.NET naming conventions for EditorConfig](#)
- [Create portable custom editor options](#)
- [.NET Compiler Platform's .editorconfig file](#)

.NET naming conventions for EditorConfig

3/1/2018 • 5 min to read • [Edit Online](#)

Naming conventions concern the naming of code elements such as classes, properties, and methods. For example, you can specify that public members must be capitalized, or that asynchronous methods must end in "Async". You can enforce these rules by specifying them in an [.editorconfig file](#). Naming rule violations appear either in the Error List or as a suggestion under the name, depending on the severity you choose for your rule. There is no need to build the project in order to see violations.

Naming conventions should be ordered from most-specific to least-specific in the .editorconfig file. The first rule encountered that can be applied is the only rule that is applied.

For each naming convention, you must specify the symbols it applies to, a naming style, and a severity for enforcing the convention, using the properties described below. The order of the properties is not important.

To begin, choose a title for your naming rule that you will use in each of the properties that are needed to fully describe the rule. For example, `public_members_must_be_capitalized` is a good, descriptive name for a naming rule. We'll refer to the title you choose as **<namingRuleTitle>** in the sections that follow.

Symbols

First, identify a group of symbols to apply the naming rule to. This property has the following format:

```
dotnet_naming_rule.<namingRuleTitle>.symbols = <symbolTitle>
```

Give a name to the group of symbols by replacing the **<symbolTitle>** value with a descriptive title, for example `public_symbols`. You'll use the **<symbolTitle>** value in the three property names that describe which symbols the rule is applied to (kinds of symbol, accessibility levels, and modifiers).

Kinds of symbols

To describe the kind of symbols to apply the naming rule to, specify a property in the following format:

```
dotnet_naming_symbols.<symbolTitle>.applicable_kinds = <values>
```

The following list shows the allowable values, and you can specify multiple values by separating them with a comma.

- * (use this value to specify all symbols)
- class
- struct
- interface
- enum
- property
- method
- field
- event
- delegate
- parameter

Accessibility levels of symbols

To describe the accessibility levels of the symbols you want the naming rule to apply to, specify a property name in

the following format:

```
dotnet_naming_symbols.<symbolTitle>.applicable_accessibilities = <values>
```

The following list shows the allowable values, and you can specify multiple values by separating them with a comma.

- * (use this value to specify all accessibility levels)
- public
- internal or friend
- private
- protected
- protected_internal or protected_friend

NOTE

Do not specify an accessibility level as part of your naming convention if accessibility is not applicable to the kind of symbol you are targeting. For example, parameters do not have accessibility levels. If you specify an accessibility level for a parameter naming convention, your naming rule will not function correctly.

Symbol modifiers

To describe the modifiers of the symbols you want the naming rule to apply to, specify a property name in the following format:

```
dotnet_naming_symbols.<symbolTitle>.required_modifiers = <values>
```

The following list shows the allowable values, and you can specify multiple values by separating them with a comma.

- abstract or must_inherit
- async
- const
- readonly
- static or shared

`required_modifiers` is an optional property. If you omit this property, your naming rule will apply to all modifiers.

Style

Now that we've identified the group of symbols to apply the naming rule to, we must describe the naming style. A style can be that the name has a certain prefix or a certain suffix, or that individual words in the name are separated with a certain character. You can also specify a capitalization style. The `style` property has the following format:

```
dotnet_naming_rule.<namingRuleTitle>.style = <styleTitle>
```

Give the style a name by replacing the `<styleTitle>` value with a descriptive title, for example

`first_word_upper_case_style`. You'll use the `<styleTitle>` value in the property names that describe the naming style (prefix, suffix, word separator character, and capitalization). Use one or more of these properties to describe your style.

Require a prefix

To specify that symbol names must begin with certain characters, use this property:

```
dotnet_naming_style.<styleTitle>.required_prefix = <prefix>
```

Require a suffix

To specify that symbol names must end with certain characters, use this property:

```
dotnet_naming_style.<styleTitle>.required_suffix = <suffix>
```

Require a certain word separator

To specify that individual words in symbol names must be separated with a certain character, use this property:

```
dotnet_naming_style.<styleTitle>.word_separator = <separator character>
```

Require a capitalization style

To specify a particular capitalization style for symbol names, use this property:

```
dotnet_naming_style.<styleTitle>.capitalization = <value>
```

The allowable values for this property are:

- pascal_case
- camel_case
- first_word_upper
- all_upper
- all_lower

NOTE

You must specify a capitalization style as part of your naming style, otherwise your naming style might be ignored.

Severity

To describe the severity of a violation of your naming rule, specify a property in the following format:

```
dotnet_naming_rule.<namingRuleTitle>.severity = <value>
```

The following table shows the allowable severity values, and what they mean:

| SEVERITY | EFFECT |
|----------------|---|
| none or silent | When this style is not being followed, do not show anything to the user; however, auto-generated code follows this style. |
| suggestion | When this style is not being followed, show it to the user as a suggestion, as underlying dots on the first two characters. It has no effect at compile time. |
| warning | When this style is not being followed, show a compiler warning in the Error List. |
| error | When this style is not being followed, show a compiler error in the Error List. |

NOTE

You do not have to build your project in order to see naming rule violations. They appear as code is edited, either in the Error List or as a suggestion.

Example

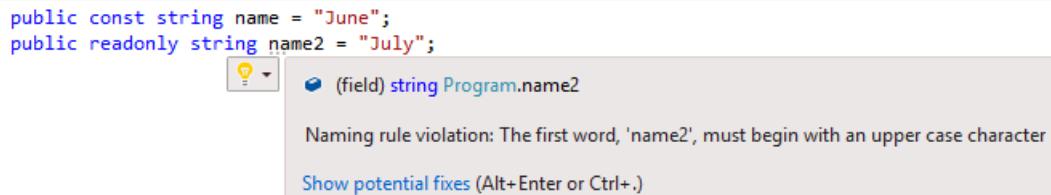
The following .editorconfig file contains a naming convention that specifies that public properties, methods, fields, events, and delegates must be capitalized. Notice that this naming convention specifies multiple kinds of symbol to apply the rule to, using a comma to separate the values.

```
# Public members must be capitalized (public_members_must_be_capitalized)
[*.{cs,vb}]
dotnet_naming_rule.public_members_must_be_capitalized.symbols      = public_symbols
dotnet_naming_symbols.public_symbols.applicable_kinds              = property,method,field,event,delegate
dotnet_naming_symbols.public_symbols.applicable_accessibilities    = public
dotnet_naming_symbols.public_symbols.required_modifiers           = readonly

dotnet_naming_rule.public_members_must_be_capitalized.style        = first_word_upper_case_style
dotnet_naming_style.first_word_upper_case_style.capitalization     = first_word_upper

dotnet_naming_rule.public_members_must_be_capitalized.severity     = suggestion
```

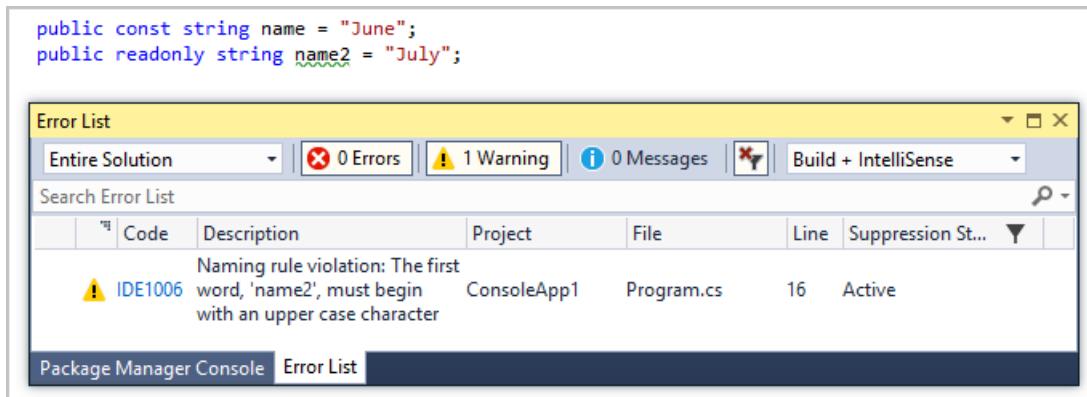
The following screenshot shows the effect of this naming convention in the Editor. Two public variables have been named without capitalization of the first letter. One is a `const`, and one is `readonly`. Since the naming rule only applies to `readonly` symbols, only the `readonly` variable shows a naming rule suggestion.



Now let's change the violation severity to `warning`:

```
dotnet_naming_rule.public_members_must_be_capitalized.severity = warning
```

If you close and reopen your code file, instead of seeing the suggestion under the name violation, you see a green squiggly, and a warning in the Error List:



See also

- [.NET language and formatting conventions](#)
- [Create portable custom editor options](#)
- [.NET Compiler Platform's .editorconfig file](#)

How to: Track Your Code by Customizing the Scrollbar

12/22/2017 • 2 min to read • [Edit Online](#)

When you are working with long code files, it can be hard to keep everything in mind. You can customize the scroll bar of the code window to give you a bird's eye view of what's happening in your code.

To show annotations on the scroll bar

1. You can set up the scroll bar to show code changes, breakpoints, errors, and bookmarks.

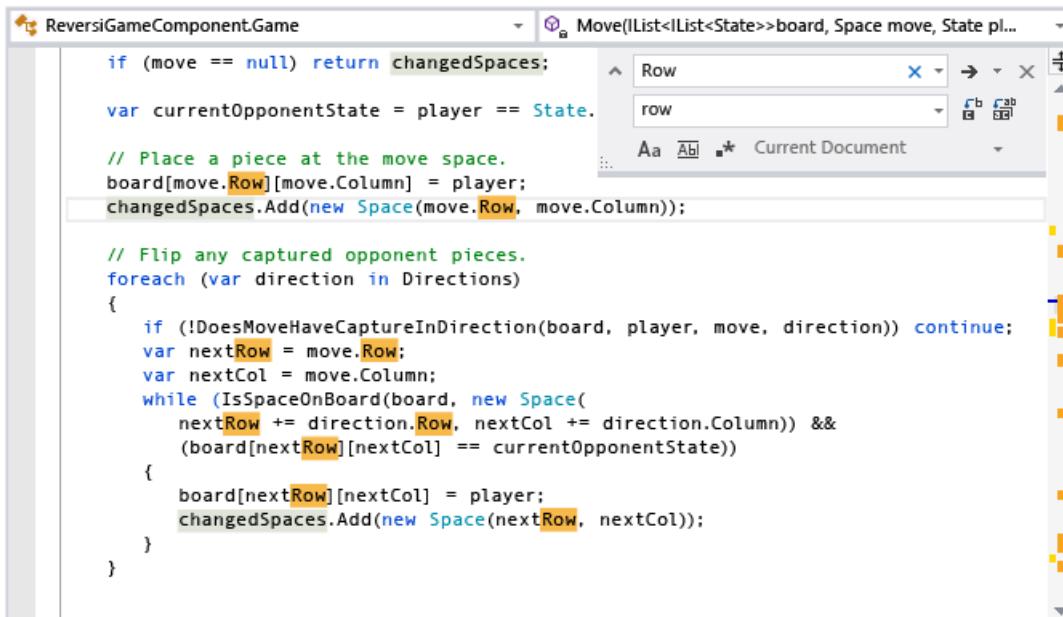
Open the **Scroll Bar** options page by choosing **Tools > Options > Text Editor > All Languages** or a specific language, or by entering **scroll bar** in the Quick Launch window.

2. Select **Show Annotations over vertical scroll bar**, then select the annotations you want to see.

The **Marks** option includes breakpoints and bookmarks.

3. Now try it out. Open a large code file and replace something that occurs in several places in the file. The scroll bar shows you the effect of the replacements, so you can back out your changes if you replaced something you shouldn't have.

Here's how the scroll bar looks after a search for a string. Notice that all instances of the string appear.



Here's the scroll bar after replacing all the instances of the string. You can see immediately that the operation caused some problems.

A screenshot of a code editor window titled "ReversiGameComponent.Game". The main area contains C# code for a "Move" method. A vertical scroll bar is visible on the right side of the code editor. The scroll bar has a light gray background with yellow vertical bars indicating the current position and range of the code. A tooltip "Row" is displayed above the scroll bar, and a dropdown menu shows "row" as the current selection.

```
if (move == null) return changedSpaces;

var currentOpponentState = player == State.One ? State.Two : State.One;

// Place a piece at the move space.
board[move.Row][move.Column] = player;
changedSpaces.Add(new Space(move.Row, move.Column));

// Flip any captured opponent pieces.
foreach (var direction in Directions)
{
    if (!DoesMoveHaveCaptureInDirection(board, player, move, direction)) continue...
    var nextRow = move.Row;
    var nextCol = move.Column;
    while (IsSpaceOnBoard(board, new Space(
        nextRow += direction.Row, nextCol += direction.Column)) &&
        (board[nextRow][nextCol] == currentOpponentState))
    {
        board[nextRow][nextCol] = player;
        changedSpaces.Add(new Space(nextRow, nextCol));
    }
}
```

To set the display mode for the scroll bar

1. The scroll bar has two modes: bar mode (the default) and map mode. Bar mode just displays annotation indicators on the scroll bar. In map mode the lines of code are represented on the scroll bar. You can choose how wide they are and whether they show the underlying code when you rest the pointer on them. When you click a location on the scroll bar, the cursor moves to that location in the code. Collapsed regions are shaded differently; they are expanded when you double-click them.

On the **Scroll Bar** options page, select either **Use Bar mode for vertical scroll bar** or **Use Map mode for vertical scroll bar**. You can choose the width in the **Source Overview** drop-down.

Here's how the search example looks when map mode is on and the width is set to Medium:

A screenshot of a code editor window titled "ReversiGameComponent.Game". The main area contains C# code for a "Move" method. A vertical scroll bar is visible on the right side of the code editor. The scroll bar has a light gray background with a grid of small squares representing the code lines. A tooltip "Row" is displayed above the scroll bar, and a dropdown menu shows "row" as the current selection.

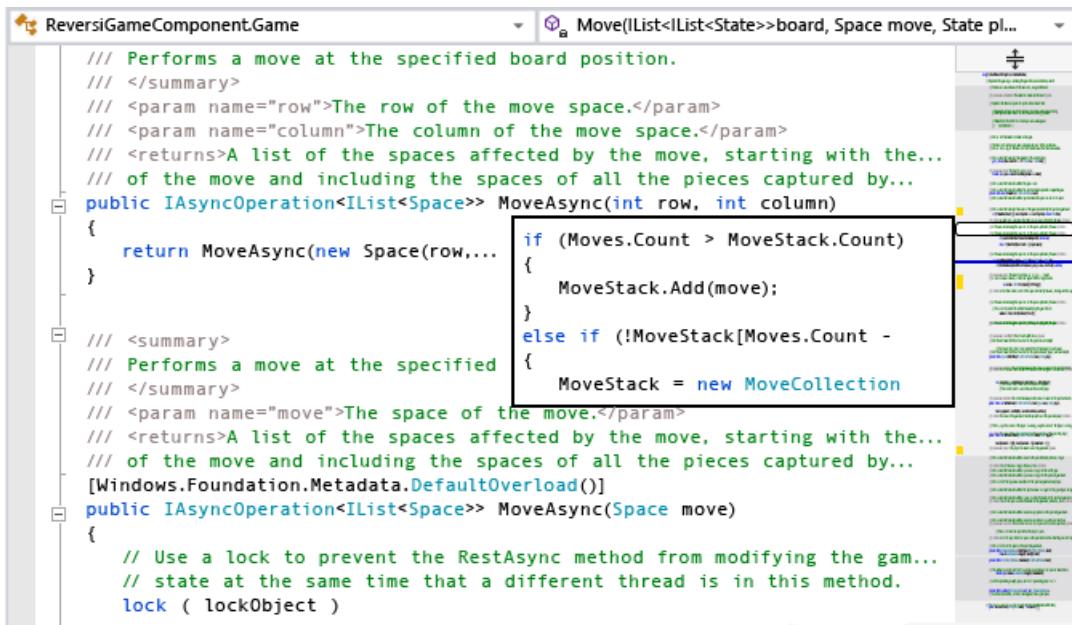
```
if (move == null) return changedSpaces;

var currentOpponentState = player == State.One ? State.Two : State.One;

// Place a piece at the move space.
board[move.Row][move.Column] = player;
changedSpaces.Add(new Space(move.Row, move.Column));

// Flip any captured opponent pieces.
foreach (var direction in Directions)
{
    if (!DoesMoveHaveCaptureInDirection(board, player, move, direction)) c...
    var nextRow = move.Row;
    var nextCol = move.Column;
    while (IsSpaceOnBoard(board, new Space(
        nextRow += direction.Row, nextCol += direction.Column)) &&
        (board[nextRow][nextCol] == currentOpponentState))
    {
        board[nextRow][nextCol] = player;
        changedSpaces.Add(new Space(nextRow, nextCol));
    }
}
```

2. In map mode, to enable previews of the code when you move the cursor up and down the scroll bar, select the **Show Preview Tooltip** option. Here's how it looks:



The screenshot shows the Visual Studio code editor with a tooltip displayed over a method call. The tooltip contains XML documentation for the `Move` method, which includes summaries, parameters, and returns sections. It also shows code snippets for performing moves and managing move stacks. The background of the tooltip is white, and the text is in a standard monospaced font used in the code editor.

```
/// Performs a move at the specified board position.
/// </summary>
/// <param name="row">The row of the move space.</param>
/// <param name="column">The column of the move space.</param>
/// <returns>A list of the spaces affected by the move, starting with the...
/// of the move and including the spaces of all the pieces captured by...
public IAsyncOperation<IList<Space>> MoveAsync(int row, int column)
{
    return MoveAsync(new Space(row, ...
}

/// <summary>
/// Performs a move at the specified
/// </summary>
/// <param name="move">The space of the move.</param>
/// <returns>A list of the spaces affected by the move, starting with the...
/// of the move and including the spaces of all the pieces captured by...
[Windows.Foundation.Metadata.DefaultOverload()]
public IAsyncOperation<IList<Space>> MoveAsync(Space move)
{
    // Use a lock to prevent the RestAsync method from modifying the gam...
    // state at the same time that a different thread is in this method.
    lock ( lockObject )
}

if (Moves.Count > MoveStack.Count)
{
    MoveStack.Add(move);
}
else if ( !MoveStack[Moves.Count - 1].IsFull )
{
    MoveStack = new MoveCollection
```

If you want to keep the map mode scrolling behavior and the preview tooltip, but don't want to see the source code overview, you can set **Source Overview** to **Off**.

See also

[Writing Code in the Editor](#)

View the structure of code

1/13/2018 • 10 min to read • [Edit Online](#)

You can examine the objects and members in Visual Studio projects, .NET Framework components, COM components, dynamic-link libraries (DLL), and type libraries (TLB).

You can also use **Solution Explorer** to browse the types and members in your projects, search for symbols, view a method's call hierarchy, find symbol references, and more without having to switch between the multiple tool windows listed previously.

If you have Visual Studio Enterprise, you can use code maps to visualize the structure of your code and its dependencies across the entire solution, and drill down to parts of the code that interest you. For more information, see [Map dependencies across your solutions](#).

NOTE

The Visual Studio edition and the settings you are using may affect the features in the IDE. They might differ from those described in this topic.

Class View (Visual Basic, C#, C++)

Class View is shown as part of **Solution Explorer** as well as in a separate window. The **Class View** window displays the elements of an application. The upper pane displays namespaces, types, interfaces, enumerations, and classes, and the lower pane displays the members that belong to the type selected in the upper pane. By using this window, you can move to member definitions in the source code (or in the **Object Browser** if the element is defined outside your solution).

You do not have to compile a project to view its elements in **Class View**. The window is refreshed as you modify the code in your project.

You can add code to your project by selecting the project node and choosing the **Add** button to open the **Add New Item** dialog box. The code is added in a separate file.

If your project is checked in to source code control, every **Class View** element displays an icon that indicates the source code status of the file. Common source code control commands such as **Check Out**, **Check In**, and **Get Latest Version** are also available on the shortcut menu for the element.

Class View toolbar

The Class View toolbar contains the following commands.

| | |
|-------------------|---|
| New Folder | Creates a virtual folder or subfolder in which you can organize frequently-used elements. They are saved in the active solution (.suo) file. After you rename or delete an element in your code, it might appear in a virtual folder as an error node. To correct this problem, delete the error node. If you renamed an element, you can move it from the project hierarchy into the folder again. |
| Back | Navigates to the previously selected item. |

| | |
|--|--|
| Forward | Navigates to the next selected item. |
| View Class Diagram (managed code projects only) | Becomes available when you select a namespace or type in Class View . When a namespace is selected, the class diagram shows all the types in it. When a type is selected, the class diagram shows only that type. |

Class View settings

The **Class View Settings** button on the toolbar has the following settings.

| | |
|--------------------------------------|---|
| Show Base Types | Base types are displayed. |
| Show Derived Types | Derived types are displayed. |
| Show Hidden Types and Members | Hidden types and members (not intended for use by clients) are displayed in light gray text. |
| Show Public Members | Public members are displayed. |
| Show Protected Members | Protected members are displayed. |
| Show Private Members | Private members are displayed. |
| Show Other Members | Other kinds of members are displayed, including internal (or Friend in Visual Basic) members. |
| Show Inherited Members | Inherited members are displayed. |
| Show Extension Methods | Extension methods are displayed. |

Class View shortcut menu

The shortcut menu in **Class View** may contain the following commands, depending on the kind of project selected.

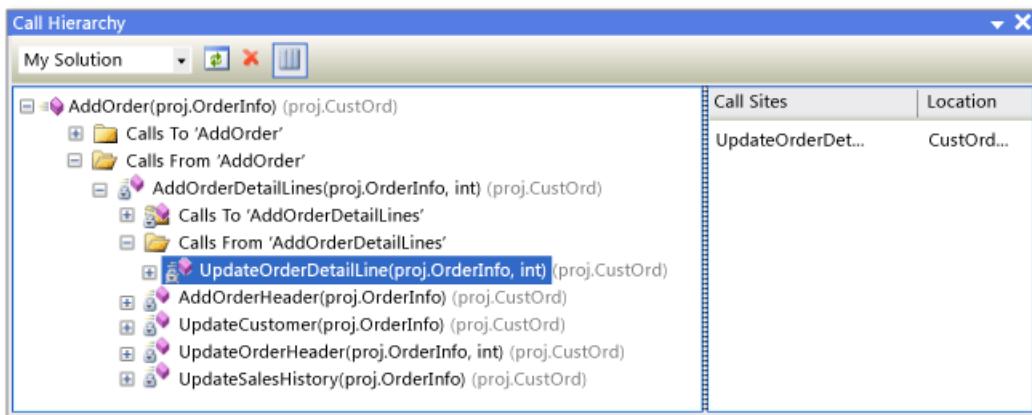
| | |
|---|---|
| Go To Definition | Finds the definition of the element in the source code, or in the Object Browser , if the element is not defined in the open project. |
| Browse Definition | Displays the selected item in the Object Browser . |
| Find All References | Finds the currently selected object item and displays the results in a Find Results window. |
| Filter To Type (managed code only) | Displays only the selected type or namespace. You can remove the filter by choosing the Clear Find (X) button next to the Find box. |
| Copy | Copies the fully qualified name of the item. |
| Sort Alphabetically | Lists types and members alphabetically by name. |

| | |
|--|--|
| Sort by Member Type | Lists types and members in order by type (such that classes precede interfaces, interfaces precede delegates, and methods precede properties). |
| Sort by Member Access | Lists types and members in order by access type, such as public or private. |
| Group by Member Type | Sorts types and members into groups by object type. |
| Go To Declaration (C++ code only) | Displays the declaration of the type or member in the source code, if available. |
| Go To Definition | Displays the definition of the type or member in the source code, if available. |
| Go To Reference | Displays a reference to the type or member in the source code, if available. |
| View Call Hierarchy | Displays the selected method in the Call Hierarchy window. |

Call Hierarchy window (Visual Basic, C#, C++)

The **Call Hierarchy** window shows where a given method (or property or constructor) is called, and lists the methods that are called from that method. You can view multiple levels of the call graph, which shows the caller/callee relationships among the methods in a specified scope.

You can display the **Call Hierarchy** window by selecting a method (or property or constructor) and then choosing **View Class Hierarchy** on the shortcut menu. The display should resemble the following picture.



By using the drop-down list on the toolbar, you can specify the scope of the hierarchy: the solution, the current project, or the current document.

The main pane displays the calls to and from the method, and the **Call Sites** pane displays the location of the selected call. For members that are virtual or abstract, an **Overrides method name** node appears. For interface members, an **Implements method name** node appears.

The **Call Hierarchy** window does not find method group references, which include places where a method is added as an event handler or is assigned to a delegate. To find these references, use the **Find All References** command.

The shortcut menu in the **Call Hierarchy** window contains the following commands.

| | |
|----------------------------|---|
| Add as New Root | Adds the selected node as a new root node. |
| Remove Root | Removes the selected root node from the tree view pane. |
| Go to Definition | Navigates to the original definition of a method. |
| Find All References | Finds in the project all the references to the selected method. |
| Copy | Copies the selected node (but not its sub-nodes). |
| Refresh | Refreshes the information. |

Object Browser

The **Object Browser** window displays descriptions of the code in your projects.

You can filter the components you want to view by using the drop-down list at the top of the window. Custom components can include managed code executables, library assemblies, type libraries, and .ocx files. It is not possible to add C++ custom components. Custom settings are saved in the Visual Studio user application directory, %APPDATA%\Microsoft\VisualStudio\15.0\ObjBrowEX.dat.

The left pane of the **Object Browser** shows assemblies. You can expand the assemblies to display the namespaces they contain, and then expand the namespaces to display the types they contain. When you select a type, its members (such as properties and methods) are listed in the right pane. The lower right pane displays detailed information about the selected item.

You can search for a specific item by using the **Search** box at the top of the window. Searches are case-insensitive. Search results are displayed in the left pane. To clear a search, choose the **Clear Search** (X) button next to the **Search** box.

The **Object Browser** keeps track of the selections you have made, and you can navigate among your selections by using the **Forward** and **Back** buttons on the toolbar.

You can use the **Object Browser** to add an assembly reference to an open solution by selecting an item (assembly, namespace, type, or member) and choosing the **Add Reference** button on the toolbar.

Object Browser settings

By using the **Object Browser Settings** button on the toolbar, you can specify one of the following views.

| | |
|---------------------------|---|
| View Namespaces | Displays namespaces rather than physical containers, in the left pane. Namespaces stored in multiple physical containers are merged. |
| View Containers | Displays physical containers rather than namespaces, in the left pane. View Namespaces and View Containers are mutually exclusive settings. |
| Show Base Types | Displays base types. |
| Show Derived Types | Displays derived types. |

| | |
|--------------------------------------|--|
| Show Hidden Types and Members | Displays hidden types and members (not intended for use by clients), in light gray text. |
| Show Public Members | Displays public members. |
| Show Protected Members | Displays protected members. |
| Show Private Members | Displays private members. |
| Show Other Members | Displays other types of members, including internal (or Friend in Visual Basic) members. |
| Show Inherited Members | Displays inherited members. |
| Show Extension Methods | Displays extension methods. |

Object Browser shortcut menu commands

The shortcut menu in the **Object Browser** may contain the following commands, depending on the kind of item selected.

| | |
|--|--|
| Browse Definition | Shows the primary node for the selected item. |
| Find All References | Finds the currently selected object item and displays the results in a Find Results window. |
| Filter To Type | Displays only the selected type or namespace. You can remove the filter by choosing the Clear Search button. |
| Copy | Copies the fully qualified name of the item. |
| Remove | If the scope is a custom component set, removes the selected component from the scope. |
| Sort Alphabetically | Lists types and members alphabetically by name. |
| Sort by Object Type | Lists types and members in order by type (such that classes precede interfaces, interfaces precede delegates, and methods precede properties). |
| Sort by Object Access | Lists types and members in order by access type, such as public or private. |
| Group by Object Type | Sorts types and members into groups by object type. |
| Go To Declaration (C++ projects only) | Displays the declaration of the type or member in the source code, if available. |
| Go To Definition | Displays the definition of the type or member in the source code, if available. |

| | |
|----------------------------|--|
| Go To Reference | Displays a reference to the type or member in the source code, if available. |
| View Call Hierarchy | Displays the selected method in the Call Hierarchy window. |

Code Definition window (C++)

The **Code Definition** window displays the definition of a selected C++ type or member in the active project. The type or member can be selected in the code editor or in a code view window.

Although this window is read-only, you can set breakpoints or bookmarks in it. To modify the displayed definition, choose **Edit Definition** on the shortcut menu. This opens the source file in the code editor and moves the insertion point to the line where the definition begins.

NOTE

Starting in Visual Studio 2015, the Code Definition window can only be used with C++ code.

Code Definition Shortcut Menu

The shortcut menu in the **Code Definition** window may contain the following commands:

| | |
|--|---|
| Quick Actions and Refactorings... | |
| Rename... | |
| Generate Graph of Include Files | |
| Peek Definition | |
| Go To Definition | Finds the definition (or definitions, for partial classes) and displays them in a Find Results window. |
| Go To Declaration | |
| Find All References | Finds the references to the type or member in the solution. |
| View Call Hierarchy | Displays the method in the Call Hierarchy window. |
| Toggle Header / Code File | |
| Run Tests | If there are unit tests in the project, runs the tests for the selected code. |
| Debug Tests | |
| Breakpoint | Inserts a breakpoint (or a tracepoint). |
| Run to Cursor | Runs the program in debug mode to the location of the cursor. |

| | |
|-------------------------|--|
| Snippet | |
| Cut, Copy, Paste | |
| Annotation | |
| Outlining | Standard outlining commands. |
| Rescan | |
| Edit Definition | Moves the insertion point to the definition in the code window. |
| Choose Encoding | Opens the Encoding window so that you can set an encoding for the file. |

Document Outline window

You can use the **Document Outline** window in conjunction with designer views, such as the designer for a XAML page or a Windows Form designer, or with HTML pages. This window displays the elements in a tree view so that you can view the logical structure of the form or page and find controls that are deeply embedded or hidden.

See also

[Class View and Object Browser icons](#)

Class View and Object Browser Icons

12/22/2017 • 1 min to read • [Edit Online](#)

Class View and the **Object Browser** display icons that represent code entities, for example, namespaces, classes, functions, and variables. The following table illustrates and describes the icons.

| ICON | DESCRIPTION | ICON | DESCRIPTION |
|------------------|----------------------|----------------------|----------------------|
| Namespace | Method or Function | Method or Function | Method or Function |
| Class | Operator | Operator | Operator |
| Interface | Property | Property | Property |
| Structure | Field or Variable | Field or Variable | Field or Variable |
| Union | Event | Event | Event |
| Enum | Constant | Constant | Constant |
| TypeDef | Enum Item | Enum Item | Enum Item |
| Module | Map Item | Map Item | Map Item |
| Extension Method | External Declaration | External Declaration | External Declaration |
| Delegate | Error | Error | Error |
| Exception | Template | Template | Template |
| Map | Unknown | Unknown | Unknown |
| Type Forwarding | | | |

Signal Icons

The following signal icons apply to all the previous icons and indicate their accessibility.

| ICON | DESCRIPTION |
|------------------|--|
| <No Signal Icon> | Public. Accessible from anywhere in this component and from any component that references it. |
| * | Protected. Accessible from the containing class or type, or those derived from the containing class or type. |
| # | Private. Accessible only in the containing class or type. |

| ICON | DESCRIPTION |
|------|--|
| ⊗ | Sealed. |
| ♥ | Friend/Internal. Accessible only from the project. |
| ▣ | Shortcut. A shortcut to the object. |

NOTE

If your project is included in a source control database, additional signal icons may be displayed to indicate source-control status, such as checked in or checked out.

See also

[Viewing the Structure of Code](#)

Designing and Viewing Classes and Types

1/26/2018 • 1 min to read • [Edit Online](#)

Design, visualize, and refactor classes and other types in your code with Class Designer in Visual Studio. Use class diagrams to create and edit classes in your C#, Visual Basic, or C++ project, understand your project structure better, or reorganize your code.

Here's more about what you can do with class diagrams:

- **Design:** Edit your project's code by editing the class diagram. Add new elements and delete unwanted ones. Your changes are reflected in code.
- **Visualize:** Understand your project's structure by viewing the classes in your project on a diagram. Customize your diagram so that you can focus on the project details that you care about the most. Save your diagram to use later for demonstration or documentation.
- **Refactor:** Override methods, rename identifiers, refactor parameters, and implement interfaces and abstract classes.

See also

[Writing Code in the Editor](#)

[Map dependencies across your solutions](#)

Working with Class Diagrams (Class Designer)

1/26/2018 • 1 min to read • [Edit Online](#)

Class diagrams help you understand the class structure of projects others have written (or that you wrote a long time ago). You can use them to customize, share and present project information with others.

The first step in presenting project information is to create a class diagram that displays what you want to show. For more information, see [Viewing Types and Relationships](#). You can create multiple class diagrams for a project that can be used to display a distinct view of the project, a chosen subset of the project's types, or a chosen subset of the members of types.

In addition to defining what each class diagram shows, you can also change the way that information is presented; for more information, see [How to: Customize Class Diagrams](#).

After you have fine-tuned one or more class diagrams, you can copy them into Microsoft Office documents and print them, or export them as image files. For more information, see [How to: Copy Class Diagram Elements to a Microsoft Office Document](#), [How to: Print Class Diagrams](#) and [How to: Export Class Diagrams As Images](#).

NOTE

Class Designer does not track the location of your source files, so changing your project structure or moving source files in the project can cause Class Designer to lose track of the type, especially the source type of a typedef, base classes, or association types. You might get an error, like **Class Designer is unable to display this type**. If you do, drag the modified or relocated source code to the class diagram again to redisplay it.

See also

[Viewing Types and Relationships](#)

[How to: Customize Class Diagrams](#)

[How to: Remove Type Shapes from Class Diagrams](#)

How to: Add Class Diagrams to Projects (Class Designer)

1/26/2018 • 1 min to read • [Edit Online](#)

To design, edit, and refactor classes and other types, add a class diagram to your C#, Visual Basic, or C++ project. To visualize different parts of the code in a project, add multiple class diagrams to the project.

You can't create class diagrams from projects that share code across multiple apps. To create UML class diagrams, see [Create UML modeling projects and diagrams](#).

To add a blank class diagram to a project

1. In Solution Explorer, right-click the project name. Then choose **Add New Item** or **Add, New Item**.
2. From the template list, choose the **Class Diagram**. For Visual C++ projects, look under **Templates**, and then under **Utility** to find this template.

The class diagram opens in Class Designer and appears as a file that has a .cd extension in Solution Explorer in the project hierarchy. Use the Class Designer toolbox to drag shapes and lines to the diagram.

3. To add multiple class diagrams, repeat the steps in this procedure.

To add a class diagram based on existing types

1. In Solution Explorer, open the class file context menu, then choose **View Class Diagram**.

-or-

In **Class View**, open the namespace or type context menu, then choose **View Class Diagram**.

To display the contents of a complete project in a class diagram

1. In Solution Explorer or Class View, right-click the project and choose **View**, then choose **View Class Diagram**.

An auto-populated Class Diagram is created.

See also

[How to: Create Types by using Class Designer](#)

[How to: View Existing Types](#)

[Designing Classes and Types](#)

[Viewing Types and Relationships](#)

[Working with Class Diagrams](#)

How to: Customize Class Diagrams (Class Designer)

12/22/2017 • 3 min to read • [Edit Online](#)

You can change the way that class diagrams display information. You can customize the whole diagram or the individual types on the design surface.

For example, you can adjust the zoom level of an entire class diagram, change how individual type members are grouped and sorted, hide or show relationships, and move individual or sets of types anywhere on the diagram.

NOTE

Customizing the way that shapes appear on the diagram doesn't change the underlying code for the types represented on the diagram.

The sections that contain type members, such as the Properties section in a class, are called compartments. You can hide or show individual compartments and type members.

In this topic

- [Zoom in and out of the class diagram](#)
- [Customize grouping and sorting of type members](#)
- [Hide compartments on a type](#)
- [Hide individual members on a type](#)
- [Show hidden compartments and members on a type](#)
- [Hide relationships](#)
- [Show hidden relationships](#)
- [Remove a shape from a class diagram](#)
- [Delete a type shape and its underlying code](#)

Zoom in and out of the class diagram

1. Open and select a class diagram file in Class Designer.
2. On the Class Designer toolbar, click the **Zoom In** or **Zoom Out** button to change the zoom level of the designer surface.

or

Specify a particular zoom value. You can use the **Zoom** drop down list or type a valid zoom level (valid range is between 10% and 400%).

NOTE

Changing the zoom level does not affect the scale of your class diagram printout.

Customize grouping and sorting of type members

1. Open and select a class diagram file in Class Designer.
2. Right-click an empty area on the design surface and point to **Group Members**.
3. Select one of the available options:
 - a. **Group by Kind** separates individual type members into a grouped list of Properties, Methods, Events, and Fields. The individual groups depend on the entities definition: for example, a class will not display any events group if there are no events yet defined for that class.
 - b. **Group by Access** separates individual type members into a grouped list based on the member's access modifiers. For example, Public and Private.
 - c. **Sort Alphabetically** displays the items that make up an entity as a single alphabetized list. The list is sorted in ascending order.

Hide compartments on a type

1. Open and select a class diagram file in the class designer.
2. Right click the member category in the type you want to customize (for example, select the **Methods** node in a class).
3. Click **Hide Compartment**.

The selected compartment disappears from the type container.

Hide individual members on a type

1. Open and select a class diagram file in Class Designer.
2. Right-click the member in the type you want to hide.
3. Click **Hide**.

The selected member disappears from the type container.

Show hidden compartments and members on a type

1. Open and select a class diagram file in Class Designer.
2. Right-click the name of the type with the hidden compartment.
3. Click **Show All Members**.

All hidden compartments and members appear in the type container.

Hide relationships

1. Open and select a class diagram file in Class Designer.
2. Right-click the association or inheritance line that you want to hide.
3. Click **Hide** for association lines, and click **Hide Inheritance Line** for inheritance lines.
4. Click **Show All Members**.

All hidden compartments and members appear in the type container.

Show hidden relationships

1. Open and select a class diagram file in Class Designer.
2. Right-click the type with the hidden association or inheritance.

Click **Show All Members** for association lines, and click **Show Base Class** or **Show Derived Classes** for inheritance lines.

Remove a shape from a class diagram

You can remove a type shape from the class diagram without affecting the type's underlying code. Removing type shapes from a class diagram affects only that diagram: the underlying code that defines the type and other diagrams that display the type are not affected.

1. On the class diagram, select the type shape you want to remove from the diagram.
2. On the **Edit** menu, choose **Remove from Diagram**.

The type shape and any lines of association or inheritance connected to the shape no longer appear on the diagram.

Delete a type shape and its underlying code

1. Right-click the shape on the design surface.
2. Select **Delete Code** from the context menu.

The shape is removed from the diagram and its underlying code is deleted from the project.

See also

- [Working with Class Diagrams](#)
[How to: Change Between Member Notation and Association Notation](#)
[How to: View Existing Types](#)
[Viewing Types and Relationships](#)

How to: Copy Class Diagram Elements to a Microsoft Office Document (Class Designer)

12/22/2017 • 1 min to read • [Edit Online](#)

You can copy shapes from a .NET class diagram (.cd file) to other documents. You'll either get a copy of the shape or its underlying code based on the kind of document where you paste it. To copy shapes from UML class diagrams in a modeling project, see [Export diagrams as images](#).

To copy a single element

- Right-click the shape and choose **Copy Image**.

To copy several elements

1. Select the shapes on the diagram that you want to copy.
2. Right-click your selection and choose **Copy Image**.

To copy all the elements in a class diagram

1. Right-click the diagram surface and choose **Select All**. (Keyboard: **Ctrl + A**)
2. On the **Edit** menu, select **Copy Image**.

You can also choose **Copy** instead of **Copy Image**. **Copy** copies the image as a regular bitmap. **Copy Image** copies the image as a vector-based image, which is better for most Office applications.

See also

[How to: Print Class Diagrams](#)

[How to: Export Class Diagrams As Images](#)

How to: Export Class Diagrams As Images (Class Designer)

12/22/2017 • 1 min to read • [Edit Online](#)

To export a class diagram that you created from code in a project, save the diagram as an image. If you want to export UML class diagrams instead, see [Export diagrams as images](#).

1. Open your class diagram (.cd) file.
2. From the **Class Diagram** menu or the diagram surface shortcut menu, choose **Export Diagram as Image**.
3. Select a diagram.
4. Select the format that you want.
5. Choose **Export** to finish exporting.

To automatically update exported images that are linked from other documents, export the diagram again in Visual Studio.

See also

[How to: Print Class Diagrams](#)

[Working with Class Diagrams](#)

How to: Print Class Diagrams (Class Designer)

12/22/2017 • 1 min to read • [Edit Online](#)

You can print a class diagram using the print feature of Visual Studio.

To print a class diagram

1. Display the class diagram. (If necessary, select the class diagram tab to display it.)
2. Click **Print** on the **File** menu.

The entire class diagram will print. Note that you may need to adjust the settings in the Page Setup Dialog box in order to print at an appropriate size.

See also

[How to: Copy Class Diagram Elements to a Microsoft Office Document](#)

[How to: Export Class Diagrams As Images](#)

How to: Add Comments to Class Diagrams (Class Designer)

12/22/2017 • 1 min to read • [Edit Online](#)

You can use comment shapes to annotate class diagrams. A comment shape has one property, **Text**, into which you can type text. Comment shapes exist only on the diagram surface and not in code.

A comment resides on the class diagram view in **Class Designer**; if you open a second class diagram onto the same project, comments you created in the first view are not visible. If you delete a diagram, all the comments it contained are also deleted.

You can resize a comment shape but you cannot change other aspects of its appearance, such as its background color, font, or font size.

To add a comment

1. Drag a comment from the **Class Designer Toolbox** onto the class diagram.
2. Click in the new comment shape on the diagram and type the text you want.

See also

[Working with Class Diagrams](#)

[Working with Class Diagrams](#)

[How to: Customize Class Diagrams](#)

How to: Create Types by using Class Designer

1/26/2018 • 1 min to read • [Edit Online](#)

To design new types for C# and Visual Basic projects, create them on a class diagram. To see existing types, see [How to: View Existing Types](#).

- [Create a new type](#)
- [Apply a custom attribute to a type](#)
- [Apply a custom attribute to a type member](#)

Create a new type

1. In the Toolbox, under Class Designer, drag one of these onto a class diagram:

- **Class or Abstract Class**
- **Enum**
- **Interface**
- **Structure (VB) or Struct (C#)**
- **Delegate**
- **Module (VB only)**

2. Name the type. Then select its access level.

3. Select the file where you want to add the initial code for the type:

- To create a new file and add it to the current project, select **Create new file** and name the file.
- To add code to an existing file, select **Add to existing file**.

If your solution has a project that shares code across multiple apps, you can add a new type to a class diagram in the app project, but only if the corresponding class file is in the same app project or is in the shared project.

4. Now add other items to define the type:

| For | Add |
|--|--|
| Classes, abstract classes, structures or structs | Methods, properties, fields, events, constructors (method), destructors (method), and constants that define the type |
| Enums | Field values that make up the enumeration |
| Interfaces | Methods, properties, and events that make up the interface |
| Delegate | Parameters that define the delegate |

| | |
|--------|--|
| Module | Methods, properties, fields, events, constructors (method), and constants that define the module |
|--------|--|

See [Creating Members](#).

Apply a custom attribute to a type

1. Click the type's shape on a class diagram.
2. In the Properties window, next to the **Custom Attributes** property for the type, click the ellipsis (...) button.
3. Add one or more custom attributes with one per line. Don't enclose them in brackets.

When you're done, the custom attributes are applied to the type.

Apply a custom attribute to a type member

1. Click the member's name in its type's shape on a class diagram, or its row in the Class Details window.
2. In the Properties window, find the member's **Custom Attributes** property.
3. Add one or more custom attributes with one per line. Don't enclose them in brackets.

When you're done, the custom attributes are applied to the type.

See also

[How to: Create Inheritance Between Types](#)

[How to: Create Associations Between Types](#) [Creating and Configuring Type Members](#)

[Working with Class Diagrams](#)

[Designing Classes and Types](#)

How to: Create Inheritance Between Types (Class Designer)

12/22/2017 • 1 min to read • [Edit Online](#)

To create an inheritance relationship between two types on a class diagram using Class Designer, connect the base type with its derived type or types. You can have an inheritance relationship between two classes, between a class and an interface, or between two interfaces.

To create an inheritance between types

1. From your project in Solution Explorer, open a class diagram (.cd) file.
If you don't have a class diagram, create it. See [How to: Add Class Diagrams to Projects](#).
2. In the **Toolbox**, under **Class Designer**, click **Inheritance**.
3. On the class diagram, draw an inheritance line between the types that you want, starting from:
 - A derived class to the base class
 - An implementing class to the implemented interface
 - An extending interface to the extended interface
4. Optionally, when you have a derived type from a generic type, click the inheritance line. In the **Properties** window, set the **Type Arguments** property to match the type that you want for the generic type.

NOTE

If a parent abstract class contains at least one abstract member, then all abstract members are implemented as non-abstract inheriting classes.

Although you can visualize existing generic types, you can't create new generic types. You also can't change the type parameters for existing generic types.

See also

- [Inheritance](#)
- [Inheritance Basics](#)
- [How to: View Inheritance Between Types](#)
- [Visual C++ Classes in Class Designer](#)

How to: Create Associations Between Types (Class Designer)

12/22/2017 • 1 min to read • [Edit Online](#)

Association lines in Class Designer show how classes in a diagram are related. An Association line represents a class that is the type of a property or field of another class in your project. Association lines are generally used to illustrate the most important relationships between classes in your project.

While you could display all fields and properties as associations, it makes more sense to show only important members as associations, depending on what you intend to emphasize in the diagram. (You can show less important members as regular members or hide them altogether.)

NOTE

Class Designer supports only unidirectional associations.

To define an association line in the Class Diagram

1. In the Toolbox, under Class Designer, select **Association**.
2. Draw a line between the two shapes you want to link with an association.

A new property is created in the first class. This property displays as an association line (not as a property within a compartment in the shape) with a default name. Its type is the shape to which the association line points.

To change the name of an association

- On the diagram surface, click the label of the association line and edit it.
- or -
- 1. Click the shape that contains the property that is shown as an association.

The shape obtains focus and its members display in the Class Details window and in the Properties window.

2. In either the Class Details window or the Properties window, edit the name field for that property and press Enter.

The name is updated in the **Class Details** window, on the association line, in the Properties window, and in code.

See also

[How to: Change Between Member Notation and Association Notation](#)

How to: Visualize a Collection Association (Class Designer)

12/22/2017 • 1 min to read • [Edit Online](#)

Properties and fields that are collections of other types can be displayed on the class diagram as a collection association. Unlike a regular association, which displays a field or property as a line linking the owning class to the field's type, a collection association is displayed as a line linking the owning class to the collected type.

To create a collection association

1. In code, create a property or field whose type is itself a strongly-typed collection.
2. In the class diagram, expand the class so that properties and fields are shown.
3. In the class, right-click the field or property and choose **Show as Collection Association**.

The property or field is shown as an association line linking to the collected type.

See also

[How to: Create Associations Between Types](#)

[Designing Classes and Types](#)

[Viewing Types and Relationships](#)

Creating and Configuring Type Members (Class Designer)

3/1/2018 • 13 min to read • [Edit Online](#)

You can add these members to types on a class diagram and configure those members in the **Class Details** window:

| TYPE | MEMBERS IT CAN CONTAIN |
|--------------------------|---|
| Class | method, property (for C# and Visual Basic), field, event (for C# and Visual Basic), constructor (method), destructor (method), constant |
| Enum | member |
| Interface | method, property, event (for C# and Visual Basic) |
| Abstract Class | method, property (for C# and Visual Basic), field, event (for C# and Visual Basic), constructor (method), destructor (method), constant |
| Structure (Struct in C#) | method, property (for C# and Visual Basic) field, event (for C# and Visual Basic), constructor (method), constant |
| Delegate | Parameter |
| Module (VB Only) | method, property, field, event, constructor, constant |

NOTE

Make property declaration more concise when a property's get and set accessors don't need additional logic by using auto-implemented properties (C# only). To show the full signature, from the **Class Diagram** menu, choose **Change Members Format, Display Full Signature**. For more information about auto-implemented properties, see [Auto-Implemented Properties](#).

Common Tasks

| TASK | SUPPORTING CONTENT |
|--|--|
| Get started: Before you create and configure type members, you must open the Class Details window. | <ul style="list-style-type: none">- Opening the Class Details Window- Class Details Usage Notes- Display of Read-Only Information- Keyboard and Mouse Shortcuts in the Class Diagram and Class Details Window |
| Create and modify type members: You can create new members, modify members, and add parameters to a method by using the Class Details window. | <ul style="list-style-type: none">- Creating Members- Modifying Type Members- Adding Parameters to Methods |

Opening the Class Details Window

By default, the Class Details Window appears automatically when you open a new class diagram (see [How to: Add Class Diagrams to Projects](#)). You can also open the Class Details window explicitly, in the following ways.

To open the Class Details window

1. Right-click on any class in the diagram to display a context menu.
2. In the context menu, click **Class Details Window**.
 - or -
 - Point to **Other Windows** on the View menu and then click **Class Details**.

Creating Members

You can create a member using any of the following tools:

- Class Designer
- Class Details window toolbar
- Class Details window

NOTE

You can also create constructors and destructors using the procedures in this section. Please bear in mind that constructors and destructors are special kinds of methods, and as such, they appear in the **Methods** compartment in class diagram shapes and in the **Methods** section of the Class Details window grid.

NOTE

The only entity you can add to a delegate is parameter. Note that the procedure entitled 'To Create a member using the Class Details Window toolbar' is not valid for this action.

To create a member using Class Designer

1. Right-click the type to which you want to add a member, point to **Add**, and then choose the type of member you want to add.

A new member signature is created and added to the type. It is given a default name that you can change in **Class Designer**, the **Class Details** window, or in the **Properties** window.

2. Optionally, specify other details about the member, such as its type.

To create a member using the Class Details Window toolbar

1. On the diagram surface, select the type to which you want to add a member.

The type obtains focus and its contents are displayed in the Class Details window.

2. In the Class Details window toolbar, click the top icon and select **New <member>** from the drop-list.

The cursor moves to the **Name** field in a row for the kind of member you want to add. For example, if you clicked **New Property**, the cursor moves to a new row in the **Properties** section of the Class Details window.

3. Type the name of the member you want to create and press Enter (or otherwise move focus, such as by pressing Tab).

A new member signature is created and added to the type. The member now exists in code and is displayed in **Class Designer**, the Class Details window, and the Properties window.

4. Optionally, specify other details about the member, such as its type.

To create a member using the Class Details Window

1. On the diagram surface, select the type to which you want to add a member.

The type obtains focus and its contents are displayed in the Class Details window.

2. In the Class Details window, in the section that contains the kind of member you want to add, click **<add member>**. For example, if you want to add a field, click **<add field>**.

3. Type the name of the member you want to create and press Enter.

A new member signature is created and added to the type. The member now exists in code and is displayed in the **Class Designer**, the Class Details window, and the Properties window.

4. Optionally, specify other details about the member, such as its type.

Note: You can also use keyboard shortcuts to create members. For more information, see [Keyboard and Mouse Shortcuts in the Class Diagram and Class Details Window](#).

Modifying Type Members

Class Designer enables you to modify the members of types that are displayed on the diagram. You can modify the members of any type displayed on a class diagram that are not read-only. You modify type members by using in-place editing on the design surface, Properties window, and the Class Details window.

All the members displayed in the Class Details window represent the members of the types on the class diagram. There are four kinds of members: methods, properties, fields, and events.

All member rows appear under headings that group the members by kind. For example, all properties appear under the heading **Properties**, which, as a node in the grid, can be collapsed or expanded.

Each member row displays the following elements:

- **Member Icon**

Each kind of member is represented by its own icon. Point the mouse at the member icon to display the member's signature. Click the member icon or the whitespace to the left of the member icon to select the row.

- **Member Name**

The **Name** column in a member row displays the name of the member. This name is also displayed in the **Name** property in the Properties window. Use this cell to change the name of any member that has read-write permissions.

If the **Name** column is too narrow to show the whole name, pointing the mouse on the member name displays the entire name.

- **Member Type**

The **MemberType** cell uses IntelliSense, which lets you select from a list of all the types available in the current project or referenced projects.

- **Member Modifier**

Change the visibility modifier of a member to either **Public** (`public`), **Private** (`private`), **Friend** (`internal`), **Protected** (`protected`), **Protected`Friend** (`protected`internal`), or **Default**.

- <add member>

The last row in the Class Details window contains the text <**add member**> in the **Name** cell. If you click this cell, you can create a new member. For more information, see [Creating Members](#).

- **Member properties in the Properties window**

The Class Details window displays a subset of the member properties that are displayed in the Properties window. Changing a property in one location will update the value of the property globally. This includes the display of its value in the other location.

- **Summary**

The **Summary** cell exposes a summary of information about the member. Click the ellipsis in the **Summary** cell to view or edit information about the **Summary**, **Return Type**, and **Remarks** for the member.

- **Hide**

When the **Hide** check box is selected, the member is not displayed in the type.

To modify a type member

1. Using Class Designer, select a type.
2. If the Class Details window is not displayed, click the **Class Details Window** button on the Class Designer toolbar.
3. Edit the values in the fields of the Class Details window grid. After each edit, press ENTER, or otherwise move focus away from the edited field, for example, by pressing TAB. Your edits reflect immediately in code.

NOTE

If you want to modify only the name of a member, you can do so by using in-place editing.

Adding Parameters to Methods

Add parameters to methods using the Class Details window. Parameters can be configured to be required or optional. Providing a value for the **Optional Default** property of a parameter instructs the designer to generate code as an optional parameter.

Parameter rows contain the following items:

- **Name**

The **Name** column in a parameter row displays the name of the parameter. This name is also displayed in the **Name** property in the Properties window. You can use this cell to change the name of any parameter with read-write permissions.

Pointing at the parameter name displays the name of the parameter if the **Name** column is too narrow to show the entire name.

- **Type**

The **Parameter Type** cell uses IntelliSense, which lets you choose from a list of all the types available in the current project or referenced projects.

- **Modifier**

The **Modifier** cell in a parameter row accepts and displays the new modifier of the parameter. To enter a new parameter modifier, use the drop-down list box to select from **None**, **ref**, **out**, or **params** in C#, and **ByVal**, **ByRef**, or **ParamArray** in VB.

- **Summary**

The **Summary** cell in a parameter row allows entering of code comments that appear in IntelliSense when entering the parameter into the code editor.

- **<add parameter>**

The last parameter row of a member contains the text in the **Name** cell. Clicking this cell lets you create a new parameter. For more information, see [To add a parameter to a method](#).

Parameter properties in the Properties window

The Properties window displays the same parameter properties displayed in the Class Details window: **Name**, **Type**, **Modifier**, **Summary**, as well as the **Optional Default** property. Changing a property in one location updates the value of the property globally, including the display of its value in the other location.

NOTE

To add a parameter to a delegate, see [Creating Members](#).

NOTE

Although a destructor is a method, it cannot have parameters.

To add a parameter to a method

1. On the diagram surface, click the type containing the method to which you want to add a parameter.

The type obtains focus and its contents display in the Class Details window.

2. In the Class Details window, expand the row of the method to which you want to add a parameter.

An indented parameter row appears, containing only a pair of parentheses and the words **<add parameter>**.

3. Click **<add parameter>**, type the name of the new parameter, and press **Enter**.

The new parameter is added to the method and the method's code. It displays in the Class Details window and the Properties window.

4. Optionally, specify other details about the parameter, such as its type.

To add an optional parameter to a method

1. On the diagram surface, click the type containing the method to which you want to add an optional parameter.

The type obtains focus and its contents display in the Class Details window.

2. In the Class Details window, expand the row of the method to which you want to add an optional parameter.

An indented parameter row appears, containing only a pair of parentheses and the words **<add parameter>**.

3. Click **<add parameter>**, type the name of the new parameter, and press **Enter**.

The new parameter is added to the method and the method's code. It displays in the Class Details window and the Properties window.

4. In the Properties window, type a value for the **Optional Default** property. Setting a parameter's Optional Default property makes that parameter optional.

NOTE

Optional parameters must be the last parameters in the parameter list.

Class Details Usage Notes

Please note the following tips for using the Class Details window.

Editable and non-editable cells

All cells in the Class Details window are editable with a few exceptions:

- The entire type is read-only, when, for example, it resides in a referenced assembly. When you select the shape in the Class Designer, the Class Details window displays its details in a read-only state.
- For indexers, the name is read-only and the rest (type, modifier, summary) are editable.
- All generics have read-only parameters in the Class Details window. To change a generic parameter, edit its source code.
- The name of the type parameter that is defined on a generic type is read-only.
- When a type's code is broken (unparsable), Class Details window displays the type's contents as read-only.

The Class Details Window and source code

- You can view source code by right-clicking a shape in the Class Details window (or the Class Designer) and then clicking View Code. The source code file opens and scrolls to the selected element.
- Changing source code is immediately reflected in the display of signature information in the Class Designer and the Class Details window. If the Class Details window is closed at the time, the new information is visible the next time you open it.
- When a type's code is broken (unparsable), Class Details window displays the type's contents as read only.

Clipboard functionality in the Class Details Window

You can copy or cut fields or rows from the Class Details window and paste them into another type. You can cut a row only if it is not read-only. When you paste the row, Class Details window assigns a new name (derived from the name of the copied row) to avoid a conflict.

Display of Read-Only Information

Class Designer and the Class Details window can display the types (and members of types) for the following:

- a project that contains a class diagram
- a project referenced from a project that contains a class diagram
- an assembly referenced from a project that contains a class diagram

In the latter two cases, the referenced entity (a type or member) is read-only in the class diagram that represents it.

An entire project or portions of it, such as individual files, may be read-only. The most common cases in which a project or one of its files is read-only are when it is under source-code control (and not checked out), it exists in an external assembly, or when the operating system considers the files to be read-only.

Source-Code Control

Because a class diagram is saved as a file in a project, you need to check out the project in order to save any

changes you make in Class Designer or the Class Details window.

Read-Only Projects

The project may be read-only for a reason other than source-code control. Closing the project displays a dialog box asking whether to overwrite the project file, discard changes (don't save) or cancel the close operation. If you choose to overwrite, project files are overwritten and made read-write. The new class diagram file is added.

Read-Only Types

If you try to save a project containing a type whose source-code file is read-only, the **Save of Read-Only File** dialog box appears, which gives you choices to save the file under a new name or new location, or to overwrite the read-only file. If you overwrite the file, the new copy is no longer read-only.

If a code file contains a syntax error, shapes displaying code in that file will be temporarily read-only until the syntax error is fixed. Shapes in this state display red text and a red icon which displays a tooltip reading "The source code file contains a parse error".

A referenced type (such as a .NET Framework type), which exists under another project node or under a referenced-assembly node, is indicated on the Class Designer design surface as read-only. A local type, which exists in the project you have open, is read-write, and its shape on the Class Designer design surface is indicated as such.

Indexers are read-write in code and the Class Details window, but the indexer name is read-only.

You cannot edit partial methods by using the Class Designer or the Class Details window; you must use the Code Editor to edit them.

You cannot edit native C++ code by using the Class Designer or the Class Details window; you must use the Code Editor to edit native C++ code.

See also

- [Viewing Types and Relationships](#)
- [Refactoring Classes and Types](#)

Keyboard and Mouse Shortcuts in the Class Diagram and Class Details Window (Class Designer)

12/22/2017 • 5 min to read • [Edit Online](#)

You can use the keyboard in addition to the mouse to perform navigational actions in Class Designer and in the **Class Details** window.

Using the Mouse in Class Designer

The following mouse actions are supported in class diagrams:

| MOUSE COMBINATION | CONTEXT | DESCRIPTION |
|----------------------|--------------------------|--|
| Double-click | Shape elements | Opens the code editor. |
| | Lollipop connector | Expand/collapse lollipop. |
| | Lollipop connector label | Invokes Show Interface command. |
| Mouse Wheel | Class diagram | Scroll vertically. |
| SHIFT + Mouse Wheel | Class diagram | Scroll horizontally. |
| CTRL + Mouse Wheel | Class diagram | Zoom. |
| CTRL + Shift + click | Class diagram | Zoom. |

Using the Mouse in the Class Details Window

Using a mouse, you can change the appearance of the Class Details window and the data it displays, in the following ways:

- Clicking any editable cell lets you edit the contents of that cell. Your changes are reflected in all places that data is stored or displayed, including in the Properties window and in source code.
- Clicking any cell of a row causes the Properties window to display the properties for the element represented by that row.
- To change the width of a column, drag the boundary on the right side of the column heading until the column is the width you want.
- You can expand or collapse compartment or property nodes by clicking the arrow symbols to the left of the row.
- The Class Details Window offers several buttons for creating new members in the current class and for navigating among the members' compartments in the Class Details Window grid. For more information, see [Class Details Window Buttons](#).

Using the Keyboard in Class Designer

The following keyboard actions are supported in class diagrams:

| KEY | CONTEXT | DESCRIPTION |
|------------------|--------------------------------------|--|
| Arrow keys | Inside type shapes | Tree-style navigation on shape contents (wrapping around shape is supported). Left and right keys expand/collapse current item if it is expandable and navigate to parent if not (see tree-view navigation for detailed behavior). |
| | Top-level shapes | Moving shapes on the diagram. |
| SHIFT+arrow keys | Inside type shapes | Building continuous selection consisting of shape elements such as members, nested types, or compartments. These shortcuts do not support wrapping around. |
| HOME | Inside type shapes | Navigate to the top-level shape title. |
| | Top-level shapes | Navigate to first shape on the diagram. |
| END | Inside type shapes | Navigate to last visible element inside the shape. |
| | Top-level shapes | Navigate to the last shape on the diagram. |
| SHIFT+HOME | Inside type shape | Selects elements within the shape starting with the current item and ending with the top-most item on the same shape. |
| SHIFT+END | Inside type shape | Same as SHIFT+HOME but in top-down direction. |
| ENTER | All contexts | Invokes default action on the shape which is also available via double-click. In most cases this is View Code but some elements define it differently (lollipops, compartment headers, lollipop labels). |
| +/- | All contexts | If currently focused element is expandable, these keys expand/collapse the element. |
| > | All contexts | On elements with children, this expands the element if it is collapsed and navigates to first child. |
| < | All contexts | Navigates to the parent element. |
| ALT+SHIFT+L | Inside type shapes + on type shapes. | Navigates to the lollipop of currently selected shape if it is present. |

| KEY | CONTEXT | DESCRIPTION |
|-------------|--------------------------------------|--|
| ALT+SHIFT+B | Inside type shapes + on type shapes. | If base type list is shown on the type shape and has more than one item, this toggles expansion state of the list (collapse/expand). |
| DELETE | On type and comment shapes | Invokes Remove from Diagram command. |
| | On everything else. | Invokes Delete from Code command (members, parameters, associations, inheritance, lollipop labels). |
| CTRL+DELETE | All contexts | Invokes Delete from Code command on selection. |
| TAB | All contexts | Navigates to next child within the same parent (supports wrapping). |
| SHIFT+TAB | All contexts | Navigates to previous child within the same parent (supports wrapping). |
| SPACE | All contexts | Toggles selection on the current element. |

Using the Keyboard in the Class Details Window

NOTE

The following key bindings were chosen to specifically mimic the experience of typing code.

Use the following keys to navigate the Class Details window:

| Key | Result |
|--------|---|
| , | If the cursor is in a parameter row, typing a comma moves the cursor to the Name field of the next parameter. If the cursor is in the last parameter row of a method, it moves the cursor to the <add parameter> field, which you can use to create a new parameter. If the cursor is elsewhere in the Class Details Window, typing a comma literally adds a comma in the current field. |
| ;) | Move the cursor to the Name field of the next member row in the Class Details Window grid. |

| | |
|-------------------------|---|
| Tab | <p>Moves the cursor to the next field, first moving left to right and then top to bottom. If the cursor is moving from a field in which you have typed text, Class Details Window processes that text and stores it if it does not produce an error.</p> <p>If the cursor is on an empty field such as <add parameter>, Tab moves it to the first field of the next row.</p> |
| <space> | <p>Moves the cursor to the next field, first moving left to right and then top to bottom. If the cursor is on an empty field such as <add parameter>, it moves to the first field of the next row. Note that <space> typed immediately after a comma is ignored.</p> <p>If the cursor is in the Summary field, typing a space adds a space character.</p> <p>If the cursor is in the Hide column of a given row, typing a space toggles the value of the Hide checkbox.</p> |
| CTRL+Tab | Switch to another document window. For example, switch from the Class Details Window to an open code file. |
| ESC (Escape) | If you have begun to type text in a field, pressing ESC acts as an undo key, reverting the field's contents to its previous value. If the Class Details Window has general focus, but no specific cell has focus, pressing ESC moves focus away from the Class Details Window. |
| Up arrow and down arrow | These keys move the cursor from row to row vertically in the Class Details Window grid. |
| Left arrow | If the cursor is in the Name column, pressing the left arrow collapses the current node in the hierarchy (if it is open). |
| Right arrow | If the cursor is in the Name column, pressing the right arrow expands the current node in the hierarchy (if it is collapsed). |

See also

[Creating and Configuring Type Members](#)

Viewing Types and Relationships (Class Designer)

1/26/2018 • 1 min to read • [Edit Online](#)

Class Designer uses class diagrams to show you the details of types, for example, their constituent members, and the relationships that they share. The visualization of these entities is actually a dynamic view into the code. This means that you can edit types on the designer and then see your edits reflected in the source code of the entity. Similarly, the class diagram is kept synchronized with changes you make to entities in code.

NOTE

If your project contains a class diagram and if your project references a type that is located in another project, the class diagram does not show the referenced type until you build the project for that type. Likewise, the diagram does not display changes to the code of the external entity until you rebuild the project for that entity.

See also

- [Designing Classes and Types](#)
- [Refactoring Classes and Types](#)
- [How to: Customize Class Diagrams](#)
- [Working with Class Diagrams](#)

How to: View Existing Types (Class Designer)

1/26/2018 • 1 min to read • [Edit Online](#)

To see an existing type and its members, add its shape to a class diagram.

You can see local and referenced types. A local type exists in the currently open project and is read/write. A referenced type exists in another project or in a referenced assembly and is read-only.

To design new types on class diagrams, see [How to: Create Types by using Class Designer](#).

To see types in a project on a class diagram

1. From a project in Solution Explorer, open an existing class diagram (.cd) file. Or if no class diagram exists, add a new class diagram to the project. See [How to: Add Class Diagrams to Projects](#).
2. From the project in Solution Explorer, drag a source code file to the class diagram.

WARNING

If your solution has a project that shares code across multiple apps, you can drag files or code to a class diagram only from these sources:

- The app project that contains the diagram
 - A shared project that was imported by the app project
 - A referenced project
 - An assembly

Shapes representing the types defined in the source code file appear on the diagram at the position where you dragged the file.

You can also view types in the project by dragging one or more types from the project node in Class View to the class diagram.

TIP

If Class View is not open, open Class View from the **View** menu.

To display types at default locations on the diagram, select one or more types in Class View, right-click the selected types, and choose **View Class Diagram**.

NOTE

If a closed class diagram containing the type already exists in the project, the class diagram opens to display the type shape. However, if no class diagram containing the type exists in the project, Class Designer creates a new class diagram in the project and opens it to display the type.

When you first display a type on the diagram, its shape appears collapsed by default. You can expand the shape to view its contents.

To display the contents of a project in a class diagram

1. In Solution Explorer or Class View, right-click the project and choose **View**, then choose **View Class Diagram**.

An auto-populated Class Diagram is created.

See also

[How to: View Inheritance Between Types](#)

[How to: Customize Class Diagrams](#)

[Viewing Types and Relationships](#)

How to: View Inheritance Between Types (Class Designer)

12/22/2017 • 1 min to read • [Edit Online](#)

You can find the inheritance relationship, if it exists, between a base type and its derived types on a class diagram in Class Designer. To create an inheritance relationship, if none exist, between two types, see [How to: Create Inheritance Between Types](#).

To find the base type

1. On the class diagram, click the type for which you want to see the base class or interface.
2. On the **Class Diagram** menu, choose **Show Base Class** or **Show Base Interfaces**.

The type's base class or interface appears selected on the diagram. Any hidden inheritance lines now appear between the two shapes.

You can also right-click the type whose base type you want to display, and choose **Show Base Class** or **Show Base Interfaces**.

To find the derived types

1. On the class diagram, click the type for which you want to see the derived classes or interfaces.
2. On the **Class Diagram** menu, choose **Show Derived Classes** or **Show Derived Interfaces**.

The type's derived classes or interfaces appear on the diagram. Any hidden inheritance lines now appear between the shapes.

You can also right-click the type for which you want to see its derived types, and choose **Show Derived Classes** or **Show Derived Interfaces**.

See also

[How to: Create Associations Between Types](#)

[Viewing Types and Relationships](#)

How to: Change Between Member Notation and Association Notation (Class Designer)

12/22/2017 • 1 min to read • [Edit Online](#)

In Class Designer, you can change the way the class diagram represents an association relationship between two types from member notation to association notation and vice versa. Members displayed as association lines often provide a useful visualization of how types are related.

NOTE

Association relationships can be represented as a member property or field. To change member notation to association notation, one type must have a member of another type. To change association notation to member notation, the two types must be connected by an association line. For more information, see [How to: Create Associations Between Types](#). If your project contains multiple class diagrams, changes that you make to the way a diagram displays association relationships affect only that diagram. To change the way another diagram displays association relationships, open or display that diagram and perform these steps.

To change member notation to association notation

1. From the project node in Solution Explorer, open the class diagram (.cd) file.
2. In the type shape on the class diagram, right-click the member property or field representing the association, and choose **Show as Association**.

TIP

If no properties or fields are visible in the type shape, the compartments in the shape might be collapsed. To expand the type shape, double-click the compartment name or right-click the type shape, and choose **Expand**.

The member disappears from the compartment in the type shape and an association line appears to connect the two types. The association line is labeled with the name of the property or field.

To change association notation to member notation

- On the class diagram, right-click the association line, and choose **Show as Property** or **Show as Field** as appropriate.

The association line disappears, and the property displays in the appropriate compartment within its type shape on the diagram.

See also

- [How to: Create Inheritance Between Types](#)
- [How to: View Inheritance Between Types](#)
- [Viewing Types and Relationships](#)
- [How to: Visualize a Collection Association](#)

Refactoring Classes and Types (Class Designer)

1/26/2018 • 3 min to read • [Edit Online](#)

When you refactor code, you make it easier to understand, maintain, and more efficient by changing its internal structure and how its objects are designed, not its external behavior. Use Class Designer and the Class Details window to reduce the work that you have to do and the chance of introducing bugs when you refactor C#, Visual Basic, or C++ code in your Visual Studio project.

NOTE

The files of a project might be read-only because the project is under source-code control and is not checked out, it is a referenced project, or its files are marked as read-only on disk. When you work in a project in one of these states, you will be presented with various ways to save your work depending on the project's state. This applies to refactoring code and also to code that you change in another way, such as directly editing it.

Common Tasks

| TASK | SUPPORTING CONTENT |
|--|---|
| Refactoring classes: You can use refactoring operations to split a class into partial classes or to implement an abstract base class. | - How to: Split a Class into Partial Classes |
| Working with interfaces: In Class Designer, you can implement an interface on the class diagram by connecting it to a class that provides code for the interface methods. | - How to: Implement an Interface |
| Refactoring types, type members, and parameters: By using Class Designer, you can rename types, override type members, or move them from one type to another. You can also create nullable types. | - Renaming Types and Type Members - Moving Type Members from One Type to Another - How to: Create a Nullable Type |

Renaming Types and Type Members

In Class Designer, you can rename a type or a member of a type on the class diagram or in the Properties window. In the Class Details window, you can change the name of a member but not a type. Renaming a type or type member propagates to all windows and code locations where the old name appeared.

To rename a name in the Class Designer

1. On the class diagram, select the type or member and click on the name.

The name of the member becomes editable.

2. Type the new name for the type or type member

To rename a name in the Class Details Window

1. To display the Class Details window, right-click the type or type member and then click **Class Details**.

The Class Details window appears.

2. In the **Name** column, change the name of the type member

3. To move focus away from the cell, press the **ENTER** key or click away from the cell.

NOTE

In the Class Details window, you can change the name of a member but not a type.

To rename a name in the Properties window

1. On the class diagram or the Class Details window, right-click the type or member and then click **Properties**.

The Properties window appears and displays properties for the type or type member.

2. In the **Name** property, change the name of the type or type member.

The new name propagates to all windows and code locations in the current project where the old name appeared.

Moving Type Members from One Type to Another

Using **Class Designer**, you can move a type member from one type to another type, if both are visible in the current class diagram.

To move a type member from one type to another

1. In a type that is visible on the design surface, right-click the member you want to move to another type, and then click **Cut**.
2. Right-click the destination type and then click **Paste**.

The property is removed from the source type and appears in the destination type.

See also

[Viewing Types and Relationships](#)

[Designing Classes and Types](#)

How to: Implement an Interface (Class Designer)

1/26/2018 • 1 min to read • [Edit Online](#)

In Class Designer, you can implement an interface on the class diagram by connecting it to a class that provides code for the interface methods. Class Designer generates an interface implementation and displays the relationship between the interface and the class as an inheritance relationship. You can implement an interface by drawing an inheritance line between the interface and the class or by dragging the interface from Class View.

TIP

You can create interfaces the same way you create other types. If the interface exists but does not appear on the class diagram, then first display it. For more information, see [How to: Create Types by using Class Designer](#) and [How to: View Existing Types](#).

To implement an interface by drawing an inheritance line

1. On the class diagram, display the interface and the class that will implement the interface.
2. Draw an inheritance line from the class and the interface.

A lollipop appears attached to the class and a label with the interface name identifies the inheritance relationship. Visual Studio generates stubs for all interface members.

For more information, see [How to: Create Inheritance Between Types](#).

To implement an interface from the Class View window

1. On the class diagram, display the class that you want to implement the interface.
2. Open Class View and locate the interface.

TIP

If Class View is not open, open Class View from the **View** menu.

3. Drag the interface node to the class shape on the diagram.

A lollipop appears attached to the class and a label with the interface name identifies the inheritance relationship. Visual Studio generates stubs for all interface members; at this point, the interface is implemented.

See also

- [How to: Create Types by using Class Designer](#)
- [How to: View Existing Types](#)
- [How to: Create Inheritance Between Types](#)
- [Refactoring Classes and Types](#)

How to: Split a Class into Partial Classes (Class Designer)

1/26/2018 • 2 min to read • [Edit Online](#)

You can divide the declaration of a class or structure among several declarations by using the `Partial` keyword in Visual Basic or the `partial` keyword in C#. You can use as many partial declarations as you want, in as many different source files as you want, or in one source file. However, all the declarations must be in the same assembly and the same namespace.

Partial classes are useful in several situations. For example, when you are working on large projects, separating a class into more than one file enables more than one programmer to work on it at the same time. When you are working with code that Visual Studio generates, you can change the class without having to re-create the source file. (Examples of code that Visual Studio generates include Windows Forms and Web Service wrapper code.) You can thus create code that uses auto-generated classes without having to modify the file that Visual Studio creates.

There are two kinds of partial methods. In C#, they are called declaring and implementing; in Visual Basic, they are called declaration and implementation.

Class Designer supports partial classes and methods. The type shape in the class diagram refers to a single declaration location for the partial class. If the partial class is defined in multiple files, you can specify which declaration location Class Designer will use by setting the **New Member Location** property in the **Properties** window. That is, when you double-click a class shape, Class Designer goes to the source file that contains the class declaration identified by the **New Member Location** property. When you double-click a partial method in a class shape, Class Designer goes to the partial method declaration. Also, in the **Properties** window, the **File Name** property refers to the declaration location. For partial classes, **File Name** lists all of the files that contain declaration and implementation code for that class. However, for partial methods, **File Name** lists only the file that contains the partial method declaration.

The following examples split the definition of class `Employee` into two declarations, each of which defines a different procedure. The two partial definitions in the examples could be in one source file or in two different source files.

NOTE

Visual Basic uses partial-class definitions to separate Visual Studio-generated code from user-authored code. The code is separated into discrete source files. For example, the **Windows Form Designer** defines partial classes for controls such as `Form`. You should not modify the generated code in these controls.

For more information about partial types in Visual Basic, see [Partial](#).

Example

To split a class definition in Visual Basic, use the `Partial` keyword, as shown in the following example.

```
' First part of class definition.  
Partial Public Class Employee  
    Public Sub CalculateWorkHours()  
    End Sub  
End Class  
  
' Second part of class definition.  
Partial Public Class Employee  
    Public Sub CalculateTaxes()  
    End Sub  
End Class
```

Example

To split a class definition in C#, use the `partial` keyword, as shown in the following example.

```
// First part of class definition.  
public partial class Employee  
{  
    public void CalculateWorkHours()  
    {  
    }  
}  
  
// Second part of class definition.  
public partial class Employee  
{  
    public void CalculateTaxes()  
    {  
    }  
}
```

See also

[Partial Classes and Methods](#)

[partial \(Type\)](#)

[partial \(Method\) \(C# Reference\)](#)

[Partial](#)

How to: Create a Nullable Type (Class Designer)

1/26/2018 • 3 min to read • [Edit Online](#)

Certain value types do not always have (or need) a defined value. This is common practice in databases, where some fields might not be assigned any value. For example, you might assign a null value to a database field to signify that it has not yet been assigned a value.

A *nullable type* is a value type that you extend so that it takes the typical range of values for that type and also a null value. For example, a nullable of `Int32`, also denoted as `Nullable<Int32>`, can be assigned any value from -2147483648 to 2147483647, or it can be assigned a null value. A `Nullable<bool>` can be assigned the values `True`, `False`, or null (no value at all).

Nullable types are instances of the `Nullable<T>` structure. Each instance of a nullable type has two public read-only properties, `HasValue` and `Value`:

- `HasValue` is of type `bool` and indicates whether the variable contains a defined value. `True` means that the variable contains a non-null value. You can test for a defined value by using a statement such as `if (x.HasValue)` or `if (y != null)`.
- `Value` is of the same type as the underlying type. If `HasValue` is `True`, `Value` contains a meaningful value. If `HasValue` is `False`, accessing `Value` will throw an invalid operation exception.

By default, when you declare a variable as a nullable type, it has no defined value (`HasValue` is `False`), other than the default value of its underlying value type.

Class Designer displays a nullable type just as it displays its underlying type.

For more information about nullable types in C#, see [Nullable Types](#). For more information about nullable types in Visual Basic, see [Nullable Value Types](#).

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

To add a nullable type by using the Class Designer

1. In the Class Diagram, expand an existing class or create a new class.
2. To add a class to the project, on the **Class Diagram** menu, click **Add**, and then click **Add Class**.
3. To expand the class shape, on the **Class Diagram** menu, click **Expand**.
4. Select the class shape. On the **Class Diagram** menu, click **Add**, and then click **Field**. A new field that has the default name **Field** will appear in the class shape and also in the **Class Details** window.
5. In the **Name** column of the **Class Details** window (or in the class shape itself), change the name of the new field to a valid and meaningful name.
6. In the **Type** column of the **Class Details** window, declare the type as a nullable type, as shown in the following code:

To add a nullable type by using the Code Editor

1. Add a class to the project. Select the project node in **Solution Explorer**, and, on the **Project** menu, click **Add Class**.
2. In the .cs or .vb file for the new class, add one or more nullable types in the new class to the class declaration.
3. From Class View, drag the new class icon to the Class Designer design surface. A class shape appears in the class diagram.
4. Expand the details for the class shape and move the mouse pointer over the class members. The tooltip displays the declaration of each member.
5. Right-click the class shape and click **Class Details**. You can view or modify the new type's properties in the **Class Details** window.

See also

[Nullable<T>](#)

[Nullable Types](#)

[Using Nullable Types](#)

[How to: Identify a Nullable Type](#)

[Nullable Value Types](#)

Working with Visual C++ Code (Class Designer)

12/22/2017 • 5 min to read • [Edit Online](#)

Class Designer displays a visual design surface called a *class diagram* that provides a visual representation of the code elements in your project. You can use class diagrams to design and visualize classes and other types in a project.

Class Designer supports the following C++ code elements:

- Class (resembles a managed class shape, except that it can have multiple inheritance relationships)
- Anonymous class (displays Class View's generated name for the anonymous type)
- Template class
- Struct
- Enum
- Macro (displays the post-processed view of the macro)
- Typedef

NOTE

This is not the same as the UML class diagram, which you can create in a Modeling Project. For more information, see [UML Class Diagrams: Reference](#).

Troubleshooting Type Resolution and Display Issues

Location of Source Files

Class Designer does not keep track of the location of source files. Therefore, if you modify your project structure or move source files in your project, Class Designer can lose track of the type (especially the source type of a typedef, base classes, or association types). You might receive an error such as **Class Designer is unable to display this type**. If you do, drag the modified or relocated source code to the class diagram again to redisplay it.

Update and Performance Issues

For Visual C++ projects, it might take 30 to 60 seconds for a change in the source file to appear in the class diagram. This delay might also cause Class Designer to throw the error **No types were found in the selection**. If you receive an error such as this, click **Cancel** in the error message and wait for the code element to appear in Class View. After you do this, Class Designer should be able to display the type.

If a class diagram does not update with changes you have made in the code, you might need to close the diagram and open it again.

Type Resolution Issues

Class Designer might not be able to resolve types for the following reasons:

- The type is in a project or assembly that is not referenced from the project that contains the class diagram.
To correct this error, add a reference to the project or assembly that contains the type. For more information, see [Managing references in a project](#).
- The type is not in the correct scope, so Class Designer cannot locate it. Ensure that the code is not missing a

`using`, `imports`, or `#include` statement. Also make sure that you have not moved the type (or a related type) out of the namespace in which it was originally located.

- The type does not exist (or has been commented out). To correct this error, make sure that you have not commented out or deleted the type.
- The type is located in a library referenced by an `#import` directive. A possible workaround is to manually add the generated code (the .tlh file) to an `#include` directive into the header file.

The error you are most likely to see for a type resolution issue is **Code could not be found for one or more shapes in class diagram '<element>'**. This error message does not necessarily indicate that your code is in error. It indicates only that class designer was unable to display your code. Try the following measures.

- Ensure that the type exists. Ensure that you have not unintentionally commented out or deleted the source code.
- Ensure that Class Designer supports the type that you entered. See [Limitations for C++ Code Elements](#).
- Try to resolve the type. The type might be in a project or assembly that is not referenced from the project that contains the class diagram. To correct this error, add a reference to the project or assembly that contains the type. For more information, see [Managing references in a project](#).
- Ensure that the type is in the correct scope so that Class Designer can locate it. Make sure that the code is not missing a `using`, `imports`, or `#include` statement. Also make sure that you have not moved the type (or a related type) out of the namespace in which it was originally located.

Troubleshooting Other Error Messages

You can find assistance with troubleshooting errors and warnings in the Microsoft Developer Network (MSDN) public forums. See the [Visual Studio Class Designer Forum](#).

Limitations for C++ Code Elements

- When a Visual C++ project is loaded, Class Designer functions in a read-only manner. You can change the class diagram, but you cannot save changes from the class diagram back to the source code.
- Class Designer supports only native C++ semantics. For Visual C++ projects that are compiled into managed code, Class Designer will only visualize code elements that are native types. Therefore, you can add a class diagram to a project, but Class Designer will not allow you to visualize elements in which the `IsManaged` property is set to `true` (that is, value types and reference types).
- For Visual C++ projects, the Class Designer reads only the definition of the type. For example, assume that you define a type in a header (.h) file and define its members in an implementation (.cpp) file. If you invoke "View Class Diagram" on the implementation (.cpp) file, Class Designer displays nothing. As another example, if you invoke "View Class Diagram" on a .cpp file that uses an `#include` statement to include other files but does not contain any actual class definitions, Class Designer again displays nothing.
- IDL (.idl) files, which define COM interfaces and type libraries, do not display in diagrams unless they are compiled to native C++ code.
- Class Designer does not support global functions and variables.
- Class Designer does not support unions. This is a special type of class in which the memory allocated is only the amount necessary for the union's largest data member.
- Class Designer does not display basic data types such as `int` and `char`.
- Class Designer does not display types that are defined outside the current project if the project does not have correct references to those types.

- Class Designer can display nested types but not the relationships between a nested type and other types.
- Class Designer cannot display types that are void or that derive from a void type.

See also

[Designing and Viewing Classes and Types](#)

[Working with Class Diagrams](#)

[Designing Classes and Types](#)

[Additional Information About Class Designer Errors](#)

[Visual C++ Classes in Class Designer](#)

[Visual C++ Structures in Class Designer](#)

[Visual C++ Enumerations in Class Designer](#)

[Visual C++ Typedefs in Class Designer](#)

Visual C++ Classes in Class Designer

1/26/2018 • 5 min to read • [Edit Online](#)

Class Designer supports C++ classes and visualizes native C++ classes in the same way as Visual Basic and C# class shapes, except that C++ classes can have multiple inheritance relationships. You can expand the class shape to show more fields and methods in the class or collapse it to conserve space.

NOTE

Class Designer does not support unions (a special type of class in which the memory allocated is only the amount necessary for the union's largest data member).

Simple Inheritance

When you drag more than one class onto a class diagram, and the classes have a class inheritance relationship, an arrow connects them. The arrow points in the direction of the base class. For example, when the following classes are displayed in a class diagram, an arrow connects them, pointing from B to A:

```
class A {};
class B : A {};
```

You can also drag only class B to the class diagram, right-click the class shape for B, and then click **Show Base Classes**. This displays its base class: A.

Multiple Inheritance

Class Designer supports the visualization of multiple-class inheritance relationships. *Multiple inheritance* is used when a derived class has attributes of more than one base class. Following is an example of multiple inheritance:

```
class Bird {};
class Swimmer {};
class Penguin : public Bird, public Swimmer {};
```

When you drag more than one class onto the class diagram, and the classes have a multiple-class inheritance relationship, an arrow connects them. The arrow points in the direction of the base classes.

Right-clicking a class shape and then clicking **Show Base Classes** displays the base classes for the selected class.

NOTE

The **Show Derived Classes** command is not supported for C++ code. You can display derived classes by going to Class View, expanding the type node, expanding the **Derived Types** subfolder, and then dragging those types onto the class diagram.

For more information about multiple-class inheritance, see [Multiple Inheritance](#) and [Multiple Base Classes](#).

Abstract Classes

Class Designer supports abstract classes (also named "abstract base classes"). These are classes that you never

instantiate, but from which you can derive other classes. Using an example from "Multiple Inheritance" earlier in this document, you might instantiate the `Bird` class as individual objects as follows:

```
int main()
{
    Bird sparrow;
    Bird crow;
    Bird eagle;
}
```

However, you might not intend to instantiate the `Swimmer` class as individual objects. You might intend only to derive other types of animal classes from it, for example, `Penguin`, `Whale`, and `Fish`. In that case, you would declare the `Swimmer` class as an abstract base class.

To declare a class as abstract, you can use the `abstract` keyword. Members marked as abstract, or included in an abstract class, are virtual and must be implemented by classes that derive from the abstract class.

```
class Swimmer abstract
{
    virtual void swim();
    void dive();
};
```

You can also declare a class as abstract by including at least one pure virtual function:

```
class Swimmer
{
    virtual void swim() = 0;
    void dive();
};
```

When you display these declarations in a Class Diagram, the class name `Swimmer` and its pure virtual function `swim` are displayed in italic in an abstract class shape, together with the notation **Abstract Class**. Notice that the abstract class type shape is the same as that of a regular class, except that its border is a dotted line.

A class derived from an abstract base class must override each pure virtual function in the base class, or the derived class cannot be instantiated. So, for example, if you derive a `Fish` class from the `Swimmer` class, `Fish` must override the `swim` method:

```
class Fish : public Swimmer
{
    void swim(int speed);
};

int main()
{
    Fish guppy;
}
```

When you display this code in a Class Diagram, Class Designer draws an inheritance line from `Fish` to `Swimmer`.

Anonymous Classes

Class Designer supports anonymous classes. *Anonymous class types* are classes declared without an identifier. They cannot have a constructor or destructor, cannot be passed as arguments to functions, and cannot be returned as return values from functions. You can use an anonymous class to replace a class name with a `typedef` name, as

in the following example:

```
typedef struct
{
    unsigned x;
    unsigned y;
} POINT;
```

Structures can also be anonymous. Class Designer displays anonymous classes and structures the same as it displays the respective type. Although you can declare and display anonymous classes and structures, Class Designer will not use the tag name that you specify. It will use the name that Class View generates. The class or structure appears in Class View and Class Designer as an element called **_unnamed**.

For more information about anonymous classes, see [Anonymous Class Types](#).

Template Classes

Class Designer supports the visualization of template classes. Nested declarations are supported. The following table shows some typical declarations.

| CODE ELEMENT | CLASS DESIGNER VIEW |
|--|---------------------|
| template <class T> | A<T> |
| class A {}; | Template Class |
| template <class T, class U> | A<T, U> |
| class A {}; | Template Class |
| template <class T, int i> | A<T, i> |
| class A {}; | Template Class |
| template <class T, template <class K> class U> | A<T, U> |
| class A {}; | Template Class |

The following table shows some examples of partial specialization.

| CODE ELEMENT | CLASS DESIGNER VIEW |
|----------------------------|---------------------|
| template<class T, class U> | A<T, U> |
| class A {}; | Template Class |
| template<class T> | A<T, T> |
| class A<T, T> {}; | Template Class |
| template <class T> | A<T, int> |
| class A<T, int> {}; | Template Class |

| CODE ELEMENT | CLASS DESIGNER VIEW |
|--|--------------------------------|
| <code>template <class T1, class T2></code> | <code>A<T1*, T2*></code> |
| <code>class A<T1*, T2*> {};</code> | Template Class |

The following table shows some examples of inheritance in partial specialization.

| CODE ELEMENT | CLASS DESIGNER VIEW |
|--|----------------------------|
| <code>template <class T, class U></code> | <code>A<T, U></code> |
| <code>class A {};</code> | Template Class |
| <code>template <class TC></code> | <code>B</code> |
| <code>class A<T, int> {};</code> | Class |
| <code>class B : A<int, float></code> | (points to Class A) |
| <code>{};</code> | <code>C</code> |
| <code>class C : A<int, int></code> | Class |
| <code>{};</code> | (points to Class A) |

The following table shows some examples of partial specialization template functions.

| CODE ELEMENT | CLASS DESIGNER VIEW |
|---|--|
| <code>class A</code> | <code>A</code> |
| <code>{</code> | <code>func<T, U> (+ 1 overload)</code> |
| <code>template <class T, class U></code> | |
| <code>void func(T a, U b);</code> | |
| <code>template <class T></code> | |
| <code>void func(T a, int b);</code> | |
| <code>};</code> | |
| <code>template <class T1></code> | <code>A<T1></code> |
| <code>class A {</code> | Template Class |
| <code>template <class T2></code> | <code>B<T2></code> |
| <code>class B {};</code> | Template Class |
| <code>};</code> | (B is contained within class A under Nested Types) |
| <code>template<> template<></code> | |
| <code>class A<type>::B<type> {};</code> | |

| CODE ELEMENT | CLASS DESIGNER VIEW |
|----------------------|---------------------|
| template <class T> | A |
| class C {}; | Class |
| class A : C<int> {}; | -> C<int> |
| | C<T> |
| | Template Class |

The following table shows some examples of template inheritance.

| CODE ELEMENT | CLASS DESIGNER VIEW |
|-------------------------|--|
| template <class T> | A |
| class C {}; | Class |
| template<> | -> B |
| class C<int> { | C<int> |
| class B {}; | Class |
| } | (B is contained within class C under Nested Types) |
| class A : C<int>::B {}; | C<T> |
| | Template Class |

The following table shows some examples of canonical specialized class connection.

| CODE ELEMENT | CLASS DESIGNER VIEW |
|------------------------|---------------------|
| template <class T> | A |
| class C {}; | Class |
| template<> | -> C<int> |
| class C<int> {}; | C<int> |
| class A : C<int> {}; | Class |
| class D : C<float> {}; | C<T> |
| | Template Class |
| | D |
| | Class |
| | -> C<float> |

| CODE ELEMENT | CLASS DESIGNER VIEW |
|---|-------------------------------|
| <pre>class B { template <class T> T min (const T &a, const T &b); };</pre> | <p>B</p> <p>min <T></p> |

See also

[Working with Visual C++ Code](#)

[Classes and Structs](#)

[Anonymous Class Types](#)

[Multiple Inheritance](#)

[Multiple Base Classes](#)

[Templates](#)

Visual C++ Structures in Class Designer

12/22/2017 • 1 min to read • [Edit Online](#)

Class Designer supports C++ structures, which are declared with the keyword `struct`. Following is an example:

```
struct MyStructure
{
    char a;
    int i;
    long j;
};
```

For more information about using the `struct` type, see [struct](#).

A C++ structure shape in a class diagram looks and works like a class shape, except that the label reads **Struct** and it has square corners instead of rounded corners.

| CODE ELEMENT | CLASS DESIGNER VIEW |
|---------------------------------------|--------------------------------|
| <code>struct StructureName {};</code> | StructureName Struct |

See also

[Working with Visual C++ Code](#)

[Classes and Structs](#)

[struct](#)

Visual C++ Enumerations in Class Designer

12/22/2017 • 1 min to read • [Edit Online](#)

Class Designer supports C++ `enum` and scoped `enum class` types. Following is an example:

```
enum CardSuit {  
    Diamonds = 1,  
    Hearts = 2,  
    Clubs = 3,  
    Spades = 4  
};  
  
// or...  
enum class CardSuit {  
    Diamonds = 1,  
    Hearts = 2,  
    Clubs = 3,  
    Spades = 4  
};
```

A C++ enumeration shape in a class diagram looks and works like a structure shape, except that the label reads **Enum** or **Enum class**, it is pink instead of blue, and it has a colored border on the left and top margins. Both enumeration shapes and structure shapes have square corners.

For more information about using the `enum` type, see [Enumerations](#).

See also

[Working with Visual C++ Code](#)

[Enumerations](#)

Visual C++ Typedefs in Class Designer

1/26/2018 • 2 min to read • [Edit Online](#)

Typedef statements create one or more layers of indirection between a name and its underlying type. The Class Designer supports C++ typedef types, which are declared with the keyword `typedef`, for example:

```
typedef class coord
{
    void P(x,y);
    unsigned x;
    unsigned y;
} COORD;
```

You can then use this type to declare an instance:

```
COORD OriginPoint;
```

Although you can declare a typedef without a name, Class Designer will not use the tag name that you specify; it will use the name that Class View generates. For example, the following declaration is valid, but it appears in Class View and Class Designer as an object named `_unnamed`:

```
typedef class coord
{
    void P(x,y);
    unsigned x;
    unsigned y;
};
```

For more information about using the `typedef` type, see [Typedefs](#).

A C++ typedef shape has the shape of the type specified in the typedef. For example, if the source declares `typedef class`, the shape has rounded corners and the label **Class**. For `typedef struct`, the shape has square corners and the label **Struct**.

Classes and structures can have nested typedefs declared within them; therefore, class and structure shapes can show nested typedef declarations as nested shapes.

Typedef shapes support the **Show as Association** and **Show as Collection Association** commands on the context menu.

The following are some examples of typedef types that the Class Designer supports:

```
typedef type name
```

name : type

typedef

Draws an association line connecting to type *name*, if possible.

```
typedef void (*func)(int)
```

```
func: void (*)(int)
```

typedef

Typedef for function pointers. No association line is drawn.

Class Designer does not display a typedef if its source type is a function pointer.

```
typedef int MyInt;
class A {
    MyInt I;
};
```

MyInt: int

typedef

A

Class

Draws an association line pointing from the source type shape to the target type shape.

```
Class B {};
```



```
typedef B MyB;
```

B

Class

MyB : B

typedef

Right-clicking a typedef shape and clicking **Show As Association** displays the typedef or class and an **Alias of** line joining the two shapes (similar to an association line).

```
typedef B MyB;
```



```
typedef MyB A;
```



```
MyBar : Bar
```

typedef

Same as above.

```
Class B {};
```



```
typedef B MyB;
```



```
class A {
    MyB B;
};
```

B

Class

MyB : B

typedef

A

Class

MyB is a nested typedef shape.

```
#include <vector>

...
using namespace std;
...

typedef vector<int> MyIntVect;

vector<T> Class

MyIntVect : vector<int>
```

typedef

```
class B {};

typedef B MyB;

class A : MyB {};
```

MyB : B

typedef

-> B

B

A

Class

-> MyB

Class Designer does not support displaying this kind of relationship by using a context menu command.

```
#include <vector>

Typedef MyIntVect std::vector<int>;

Class MyVect : MyIntVect {};
```

std::vector<T>

Class

```
MyIntVect : std::vector<int>
```

typedef

MyVect

Class

-> MyIntVect

See also

[Working with Visual C++ Code](#)

[Typefdefs](#)

Additional Information About Class Designer Errors

12/22/2017 • 1 min to read • [Edit Online](#)

Class Designer does not track the location of your source files, so modifying your project structure or moving source files in the project can cause Class Designer to lose track of the type (especially the source type of a typedef, base classes, or association types). You might receive an error such as **Class Designer is unable to display this type**. If you do, drag the modified or relocated source code to the class diagram again to redisplay it.

You can find assistance with other errors and warnings in the following resources:

[Working with Visual C++ Code](#)

Includes troubleshooting information about displaying C++ in a class diagram.

[Visual Studio Class Designer Forum](#)

Provides a forum for questions about the Class Designer.

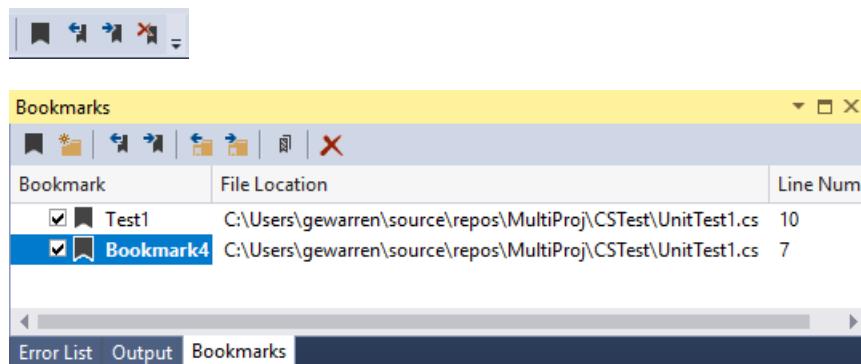
See also

[Designing and Viewing Classes and Types](#)

Set bookmarks in code

3/1/2018 • 1 min to read • [Edit Online](#)

You can use bookmarks to mark lines in your code so that you can quickly return to a specific location, or jump back and forth between locations. Bookmark commands and icons are available in two places: the **Bookmark Window** (**View > Bookmark Window**) and the text editor toolbar.



Manage bookmarks

To add a bookmark, place the cursor on the line you want to bookmark. Choose the **Toggle a bookmark** button, or press **Ctrl+K, Ctrl+K**. This adds the bookmark. If you choose the **Toggle a bookmark** button (or press **Ctrl+K, Ctrl+K**) again, the bookmark is removed.

To know at a glance what a specific bookmark is for, you can rename it in the **Bookmark Window** from the right-click or context menu. You can delete bookmarks by choosing the **Delete** button in the bookmark window.

IMPORTANT

The bookmark is set to the line number, not to the code. If you modify the code, the bookmark is retained at the line number, and does not move with the code.

You can navigate between bookmarks by using the **Next bookmark** and **Previous bookmark** buttons in the bookmark window.

You can organize bookmarks into virtual folders by choosing **New Folder** in the bookmark window and then dragging selected bookmarks into the new folder.

You can turn off bookmarks (without removing them) by choosing the **Disable All Bookmarks** button in the bookmark window. You can re-enable them by choosing the same button (which is now called **Enable All Bookmarks**).

See also

- [Writing Code in the Editor](#)

Using the Task List

12/22/2017 • 2 min to read • [Edit Online](#)

Use the **Task List** to track code comments that use tokens such as `TODO` and `HACK`, or custom tokens, and to manage shortcuts that will take you directly to a predefined location in the code. Click on the item in the list to go to its location in the source code.

The Task List window

When the **Task List** is open, it appears at the bottom of the application window.

To open the Task List

- On the **View** menu, choose **Task List** (Keyboard: **Ctrl+T**).

| Task List | | | | | |
|---|-----------------|---------------------|------|--------------|--|
| Description | Project | File | Line | Project Rank | |
| return _signInManager?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>(); ConfigureAuth(app); | | ManageController.cs | 34 | 1 | |
| | | Startup.cs | 18 | 1 | |
| | | Startup.cs | 22 | 1 | |
| TODO Add some logic here. | WebApplication1 | ManageController.cs | 21 | 1 | |
| UNDONE Check-in #42 | WebApplication1 | ManageController.cs | 26 | 1 | |
| HACK temporary workaround | WebApplication1 | ManageController.cs | 33 | 1 | |
| Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType | WebApplication1 | IdentityModels.cs | 14 | 1 | |

To change the sort order of the list

- Click the header of any column. To further refine your search results, press Shift and click a second column header.

As an alternative, on the shortcut menu, choose **Sort by**, and choose a header. To further refine your search results, press Shift and choose a second header.

To show or hide columns

- On the shortcut menu, choose **Show Columns**. Choose the columns that you want to show or hide.

To change the order of the columns

- Drag any column header to the location that you want.

User Tasks

The user task feature was removed starting in Visual Studio 2015. When you open a solution which has user task data from Visual Studio 2013 and earlier, the user task data in your .suo file will not be affected, but the user tasks will not be displayed in the task list.

If you wish to continue to access and update your user task data, you should open the project in Visual Studio 2013 and copy the content of any user tasks into your preferred project management tool (such as Team Foundation Server).

Tokens and comments

A comment in your code preceded by a comment marker and a predefined token will also appear in the **Task List**.

window. For example, the following C# comment has three distinct parts:

- The comment marker (`//`)
- The token, for example (`TODO`)
- The comment (the rest of the text)

```
// TODO: Load state from previously suspended application
```

Because `TODO` is a predefined token, this comment appears as a `TODO` task in the list.

Custom tokens

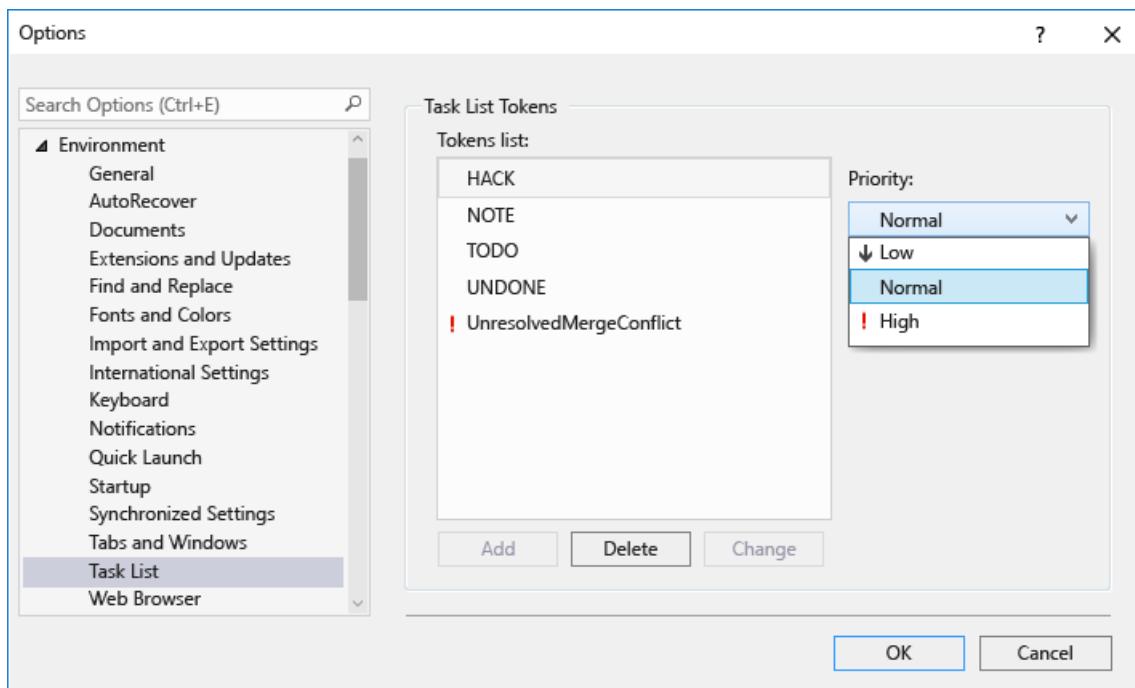
By default, Visual Studio includes the following tokens: HACK, TODO, UNDONE, NOTE. These are not case sensitive.

You can also create your own custom tokens.

To create a custom token

1. On the **Tools** menu, choose **Options**.
2. Open the **Environment** folder and then choose **Task List**.

The [Task List options page](#) is displayed.



3. In the **Tokens** category, in the **Name** text box, enter your token name, for example "BUG".
4. In the **Priority** drop-down list, choose a default priority for the new token. Choose the **Add** button.

C++ TODO comments

By default, C++ TODO comments are displayed in the **Task List** window. You can change this behavior.

To turn off C++ TODO comments

On the **Tools** menu, choose **Options > Text Editor > C/C++ > View > Enumerate Comment Tasks** and set the value to false.

Shortcuts

A *shortcut* is a bookmark in the code that is tracked in the **Task List**; it has a different icon than a regular bookmark. Double-click the shortcut in the **Task List** to go to the corresponding location in the code.



To create a shortcut

To create a shortcut, insert the pointer into the code where you want to place a shortcut. Choose **Edit > Bookmarks > Add Task List Shortcut** or press **Ctrl + K, Ctrl + H**.

To navigate through the shortcuts in the code, choose a shortcut in the list, and then choose **Next Task** or **Previous Task** from the shortcut menu.

See also

[Task List, Environment, Options Dialog Box](#)

Find code changes and other history with CodeLens

1/26/2018 • 8 min to read • [Edit Online](#)

Stay focused on your work while you find out what happened to your code - without leaving the editor. Find references and changes to your code, linked bugs, work items, code reviews, and unit tests.

NOTE

CodeLens is available only in Visual Studio Enterprise and Visual Studio Professional editions. It is not available in Visual Studio Community edition.

See where and how the individual parts of your code are used in your solution:

The screenshot shows the Visual Studio code editor with the 'CustomersControllerTest.cs' file open. A tooltip from the 'CodeLens' feature is displayed over the 'Create' method. The tooltip contains:

- A list of references: "28 : controller.Create(new Customer());" and "38 : controller.Create(null);".
- A link to "GuardHelper.cs (16,18)".
- A snippet of the 'GuardHelper.cs' code: "controller.Create(new Customer());".
- A note: "3 references | 0/2 passing | Francis Totten, 3 hours ago".
- The code for the 'Create' method:

```
public ActionResult Create(Customer customer)
{
    if (customer == null)
```

Contact your team about changes to your code without leaving the editor:

The screenshot shows the Visual Studio code editor with the 'Create' method from 'CustomersControllerTest.cs'. A tooltip is shown, containing:

- A table of recent changes:

| ID | Branch | Description |
|-----|-----------------|--|
| 600 | ⌘ FabrikamFi... | Updated UI strings for new season |
| 599 | ⌘ FabrikamFi... | Fixed special character error in Create method |
| 598 | ⌘ FabrikamFi... | Minor fix to code formatting |
| 596 | ⌘ FabrikamFi... | Added guard helper class |
| 595 | ⌘ FabrikamFi... | Minor UI updates |
- A user profile for "Francis Totten" with status "Available - Video Capable" and title "PROGRAM MANAGER".
- A list of recent commits:
 - Jamal Hartnett 5/5/2015 1
 - Francis Totten 5/1/2015 6
 - Jamal Hartnett 4/30/2015
- The code for the 'Create' method:

```
public ActionResult Create(Customer customer)
{
    if (customer == null)
```

To choose the indicators that you want to see, or to turn CodeLens off and on, go to **Tools, Options, Text Editor, All Languages, CodeLens**.

Find references to your code

You'll need:

- Visual Studio Enterprise or Visual Studio Professional
- C# or Visual Basic code

Choose the **references** indicator (**Alt + 2**). If you see **0 references**, you have no references from C# or Visual Basic code. This doesn't include references from other items such as XAML and ASPX files.

The screenshot shows the Visual Studio CodeLens interface. A tooltip for the 'Create' method in 'CustomersController.cs' lists two references: one from 'CustomersControllerTest.cs' and one from 'GuardHelper.cs'. Both references are highlighted with yellow boxes. Below the tooltip, the actual code for the 'Create' method is shown:

```
3 references 0/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items
public ActionResult Create(Customer customer)
{
    if (customer == null)
    {

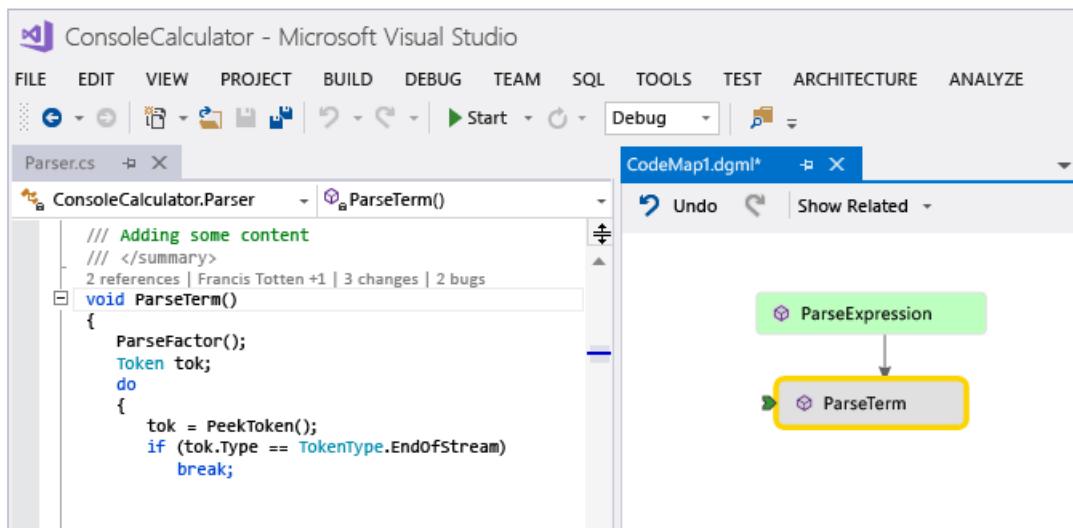
```

To view the referencing code, move your mouse on top of the reference.

The screenshot shows the Visual Studio CodeLens interface. A tooltip for the 'Create' method in 'CustomersController.cs' lists two references: one from 'CustomersControllerTest.cs' and one from 'GuardHelper.cs'. The reference from 'GuardHelper.cs' is highlighted with a yellow box. A callout arrow points from this reference to a separate window displaying the 'GuardHelper.cs' code, specifically the line 'controller.Create(new Customer());' which is also highlighted with a yellow box.

To open the file containing the reference, double-click the reference.

To see relationships between this code and its references, [create a code map](#) and choose **Show All References** in the code map shortcut menu.



Find your code's history and linked items

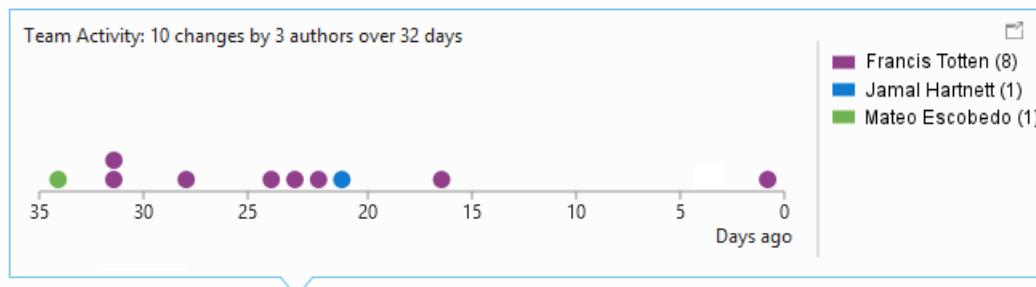
Review your code's history to find out what happened to your code. Or, review changes before they're merged into your code so you can better understand how changes in other branches might affect your code.

You'll need:

- Visual Studio Enterprise or Visual Studio Professional
- Team Foundation Server 2013 or later, Visual Studio Team Services, or Git
- [Lync 2010 or later, or Skype for Business](#), to contact your team from the code editor

For C# or Visual Basic code that's stored with Team Foundation version control (TFVC) or Git, you get CodeLens

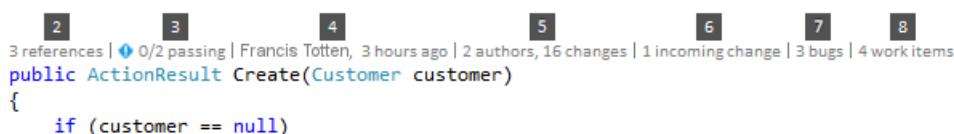
details at the class and method levels (*code-element-level* indicators). If your Git repository is hosted in TfGit, you also get links to TFS work items.



For all other types of files that you can open in the Visual Studio editor, you get CodeLens details for the entire file in one place at the bottom of the window (*file-level* indicators).



To use the keyboard to select indicators, press and hold the **ALT** key to display the related number keys.



Find changes in your code

Find who changed your C# or Visual Basic code, and the changes they made, in code-element-level indicators. This is what you see when you use Team Foundation version control (TFVC) in Team Foundation Server or Visual Studio Team Services.

| ID | Branch | Description | Author | Date |
|-----|---------------|--|----------------|------------|
| 600 | FabrikamFi... | Updated UI strings for new season | Francis Totten | 5/21/2015 |
| 599 | FabrikamFi... | Fixed special character error in Create method | Francis Totten | 5/18/2015 |
| | | Bug 165 - Fix error for special characters in username | | |
| 598 | FabrikamFi... | Minor fix to code formatting | Jamal Hartnett | 5/5/2015 1 |
| 596 | FabrikamFi... | Added guard helper class | Francis Totten | 5/1/2015 6 |
| 595 | FabrikamFi... | Minor UI updates | Jamal Hartnett | 4/30/2015 |

Show all file changes

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items

`public ActionResult Create(Customer customer)`

{

if (customer == null)

The default time period is the last 12 months. If your code is stored in Team Foundation Server, you can change this by running the **TFSConfig command** with the **CodelIndex command** and the **/indexHistoryPeriod** flag.

To see a detailed history of all the changes, including those from more than a year ago, choose **Show all file changes**.

The screenshot shows a 'Changes' window with a table of changesets. The columns are ID, Branch, Description, Author, Date, and a Local Version button. Changesets 600 and 599 are expanded to show their commit details. A 'Show all file changes' button is highlighted with a yellow box.

| ID | Branch | Description | Author | Date |
|-----|---------------|--|----------------|-----------|
| 600 | FabrikamFi... | Updated UI strings for new season | Francis Totten | 5/21/2015 |
| 599 | FabrikamFi... | Fixed special character error in Create method | Francis Totten | 5/18/2015 |
| | | Bug 165 - Fix error for special characters in username | | |
| 598 | FabrikamFi... | Minor fix to code formatting | Jamal Hartnett | 5/5/2015 |
| 596 | FabrikamFi... | Added guard helper class | Francis Totten | 5/1/2015 |
| 595 | FabrikamFi... | Minor UI updates | Jamal Hartnett | 4/30/2015 |

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items

```
public ActionResult Create(Customer customer)
{
    if (customer == null)
```

This opens the History window for the changesets.

The screenshot shows a 'History' window for 'CustomersController.cs'. It has tabs for 'Changesets' and 'Labels'. The 'Changesets' tab is selected, showing a list of changesets with columns: Changeset, Change, User, Date, Path, and Comment. Changeset 600 is selected.

| Changeset | Change | User | Date | Path | Comment |
|-----------|--------|----------------|----------------------|---------------------|---|
| 600 | edit | Francis Totten | 5/21/2015 4:55:09 AM | \$/FabrikamFiber... | Updated UI strings for new season |
| 599 | edit | Francis Totten | 5/18/2015 4:31:03 AM | \$/FabrikamFiber... | Fixed special character error in Create |
| 598 | edit | Jamal Hartnett | 5/5/2015 1:59:51 AM | \$/FabrikamFiber... | Minor fix to code formatting |
| 596 | edit | Francis Totten | 5/1/2015 6:26:31 AM | \$/FabrikamFiber... | Added guard helper class |
| 595 | edit | Jamal Hartnett | 4/30/2015 6:36:59 AM | \$/FabrikamFiber... | Minor UI updates |
| 594 | edit | Francis Totten | 4/29/2015 7:07:43 AM | \$/FabrikamFiber... | Minor fix to formatting |
| 593 | edit | Francis Totten | 4/28/2015 7:36:25 AM | \$/FabrikamFiber... | Perf review with new data store |

When your files are in a Git repository and you choose the code-element-level changes indicator, this is what you see.

The screenshot shows a 'Changes' window for a Git repository. It lists commits with their IDs, descriptions, authors, and dates. Commit 9350d639 is expanded to show its changes. A tooltip provides detailed information about this commit.

| Commit ID | Description | Author | Date |
|-----------|--|----------------|-----------|
| 9350d639 | Fixed special characters error in Create | Francis Totten | 5/18/2015 |
| | 165 - Fix error for special characters in username | | |
| f0c985ea | Minor UI presentation adjustments | Mateo Escobedo | 4/30/2015 |
| 2af030f3 | Minor updates to code formatting | Francis Totten | 5/5/2015 |
| 02a384c0 | Minor formatting updates | Francis Totten | 4/29/2015 |
| 532d3037 | Review perf with new datastore Related | Jamal Hartnett | 4/28/2015 |
| | 164 - Review performance against new data store | | |

1 reference | Francis Totten, 3 days ago | 3 authors, 10 changes | 4 bugs

```
public ActionResult Create(Customer customer)
{
    //check model state
    if (ModelState.IsValid)
    {
        this.customerRepository.InsertOrUpdate(customer);
    }
}
```

ID: 9350d639
Date: 5/18/2015 4:32:16 AM
Author: Francis Totten (francist@fabrikam.com)
Change type: Edit
Description: Fixed special characters error in Create method
Related Work Items: #165

Find changes for an entire file (except for C# and Visual Basic files) in the file-level indicators at the bottom of the window.

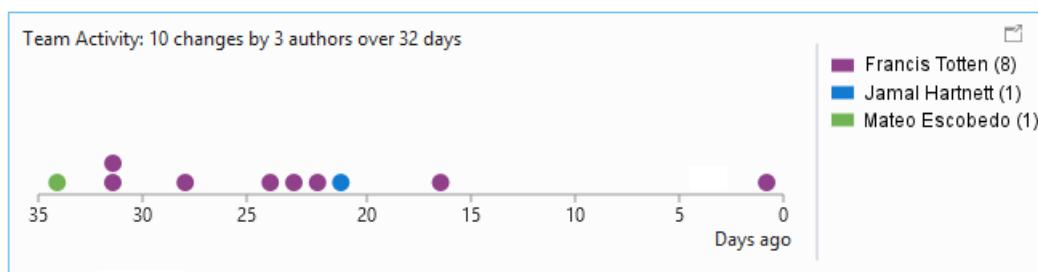
Commit ID Description Author Date

- 9350d639 Fixed special characters error in Create method... Francis Totten (francist@cr... 5/18/2015
 - 165 - Fix error for special characters in username
- f0c985ea Minor UI presentation adjustments Jamal Hartnett (jamal@fa... 4/30/2015
- ea00c408 Initial Commit Mateo Escobedo (mateo@... 4/17/2015

90 100 % < Francis Totten, 3 days ago 3 authors, 3 changes 1 work item Error List Output Ready

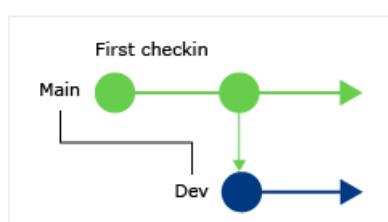
To get more details about a change, right-click that item. Depending on whether you are using TFVC or Git you get a series of options to compare the versions of the file, view details and track the changeset, get the selected version of the file, and email the author of that change. Some of these details appear in Team Explorer.

You can also see who changed your code over time. This can help you find patterns in your team's changes and assess their impact.



Find changes in your current branch

Suppose your team has multiple branches - a main branch and a child development - to reduce the risk of breaking stable code:



Find how many people changed your code and how many changes were made (**Alt + 6**) in your main branch:

ID Branch Description Author Date

- 13 FabrikamFiber.CallCenter-Main Initial checkin Local Version Francis Totten 12/17/2013

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items

```
public static void ThrowIfNullOrEmpty(string value, string name)
{
    if (string.IsNullOrEmpty(value))
        throw new ArgumentNullException(name);
```

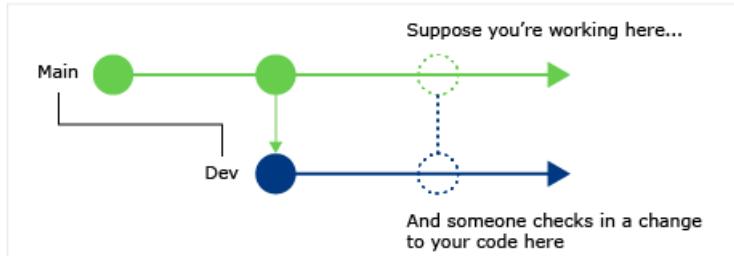
Find when your code was branched

Go to your code in the child branch, for example, the Dev branch here. Choose the changes indicator (**Alt + 6**):

| ID | Branch | Description | Author | Date |
|------|---------------------------------------|--------------------|-----------|------------|
| ▲ 15 | FabrikamFiber.CallCenter-Dev (from... | Branched from Main | Franci... | 12/17/2... |
| ▷ 13 | FabrikamFiber.CallCenter-Main | First checkin | Franci... | 12/17/2... |

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items
 □ `public static void ThrowIfNullOrEmpty(string value, string name)`
 {
 if (string.IsNullOrEmpty(value))
 throw new ArgumentNullException(name);

Find incoming changes from other branches



...like this bug fix in the Dev branch here:

| ID | Branch | Description | Author | Date |
|------|------------------------------|-----------------------|--------------|------------|
| ▲ 17 | FabrikamFiber.CallCenter-Dev | Fix bug for null name | Francis T... | 12/17/2013 |
| ▷ 15 | FabrikamFiber.CallCenter-Dev | Branched from Main | Francis T... | 12/17/2013 |

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items
 □ `public static void ThrowIfNullOrEmpty(string value, string name)`
 {
 if (string.IsNullOrEmpty(value))
 throw new ArgumentNullException(name);

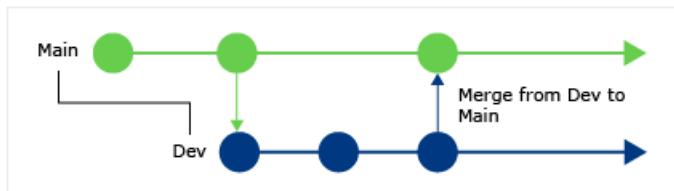
You can review this change without leaving your current branch (Main):

| ID | Branch | Description | Author | Date |
|------|-------------------------------|-----------------------|----------------|------------|
| ▲ 17 | FabrikamFiber.CallCenter-Dev | Fix bug for null name | Francis Totten | 12/17/2013 |
| ▷ 13 | FabrikamFiber.CallCenter-Main | First checkin | Francis Totten | 12/17/2013 |

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items
 □ `public static void ThrowIfNullOrEmpty(string value, string name)`
 {
 if (string.IsNullOrEmpty(value))
 throw new ArgumentNullException(name);

Find when changes got merged

So you can see which changes are included in your branch:



For example, your code in the Main branch now has the bug fix from the Dev branch:

| ID | Branch | Description | Author |
|------|--|------------------------|--------|
| ▲ 18 | FabrikamFiber.CallCenter-Main (from Fabri... | Merge from Dev to Main | Fra... |
| ▷ 17 | FabrikamFiber.CallCenter-Dev | Fix bug for null name | Fra... |
| ▷ 13 | FabrikamFiber.CallCenter-Main | Fix checkin | Fra... |

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items
 □ `public static void ThrowIfNullOrEmpty(string value, string name)`
 {
 if (string.IsNullOrEmpty(value))
 throw new ArgumentNullException(name);

Compare an incoming change with your local version (Shift + F10)

| ID | Branch | Description | Local Version |
|-------|-----------------|--|---------------|
| 600 | ⌘ FabrikamFi... | Updated UI strings for new season | Local Version |
| ▷ 599 | ⌘ FabrikamFi... | Fixed special character error in Create method | |
| 598 | ⌘ FabrikamFi... | Minor fix to code formatting | |
| 596 | ⌘ FabrikamFi... | Added guard helper class | |
| 595 | ⌘ FabrikamFi... | Minor UI updates | |
| 594 | ⌘ FabrikamFi... | Minor fix to formatting | |

Show all file changes

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change

```
public ActionResult Create(Customer customer)
{
    if (customer == null)
```

- [Compare with Local Version](#)
- [Changeset Details](#)
- [Track Changeset 600](#)
- [Hide Intermediate Branch Merges](#)
- [Send IM to Francis Totten](#)
- [Call Francis Totten](#)
- [Video Chat with Francis Totten](#)
- [Send Email to Francis Totten](#)
- [Open Contact Card](#)

You can also double-click the changeset.

What do the icons mean?

| ICON | WHERE DID THE CHANGE COME FROM? |
|------|---|
| ⌘ | The current branch |
| ⤒ | The parent branch |
| ⤓ | A child branch |
| ⤔ | A peer branch |
| ⤖ | A branch further away than a parent, child, or peer |
| ⤘ | A merge from the parent branch to a child branch |
| ⤙ | A merge from a child branch to the parent branch |
| ⤚ | A merge from an unrelated branch (baseless merge) |

Find linked work items

| ID | Type | Branch | Title | Details |
|-------|---------|-----------|--|---|
| ▲ 164 | Task | Fabrik... | Review performance against new data sto... | Description 593 - FabrikamFiber.CallCenter-Dev - Perf review with r... |
| ▲ 162 | Task | Fabrik... | Refactor for dependency injection of auth... | Description 590 - FabrikamFiber.CallCenter-Dev - Implemented dep... |
| ▲ 158 | Task | Fabrik... | Update to support category 2 accounts | Description 582 - FabrikamFiber.CallCenter-Dev - Update to support... |
| ▲ 93 | Product | Fabrik... | Application to support Fabrikam Call cent... | Francis Totten (francist@fabrikam.com) 4/20/2015 Francis Totten (francist@fabrikam.com) 4/20/2015 Francis Totten (francist@fabrikam.com) 12/16/2013 |

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs [4 work items](#)

```
public ActionResult Create(Customer customer)
{
```

```
    if (customer == null)
```

Find linked code reviews

| ID | Type | Title | Author | Date |
|----|---------------------|-------------------------------|----------------|----------|
| 32 | Code Review Request | More comments for Calculator | Jamal Hartnett | 8/4/2013 |
| 26 | Code Review Request | Added comments for Calculator | Jamal Hartnett | 8/2/2013 |

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items [2 reviews](#)

```
public ActionResult Create(Customer customer)
{
    if (customer == null)
```

Find linked bugs

| ID | Type | Branch | Title | Changeset Author | Date |
|-----|------|-----------|--|---------------------|-----------|
| 165 | Bug | Fabrik... | Fix error for special characters in username | Francis Totten (fra | 5/21/2015 |
| 160 | Bug | Fabrik... | Changes to repository code require addition | Jamal Hartnett (ja | 4/20/2015 |
| 154 | Bug | Fabrik... | UT failures for customer create | Francis Totten (fra | 4/16/2015 |

Description 599 - FabrikamFiber.CallCenter-Dev - Fixed special char:
Description 583 - FabrikamFiber.CallCenter-Dev - Updated to match
Description 579 - FabrikamFiber.CallCenter-Release - Fix test

ten, 3 hours ago | 2 authors, 16 changes | 1 incoming change [3 bugs](#) 4 work items [te\(Customer customer\)](#)

```
)  
  
ntNullException();
```

Description 599
Francis Totten (francist@fabrikam.com)
Fixed special character error in Create method
This changeset was made in \$/FabrikamFiber/
FabrikamFiber.CallCenter-Dev, the current branch.

Contact the owner of an item

| ID | Branch | Description |
|-----|-----------------|--|
| 600 | ↗ FabrikamFi... | Updated UI strings for new season |
| 599 | ↗ FabrikamFi... | Fixed special character error in Create method |
| 598 | ↗ FabrikamFi... | Bug 165 - Fix error for special characters in username |
| 596 | ↗ FabrikamFi... | Minor fix to code formatting |
| 595 | ↗ FabrikamFi... | Added guard helper class |
| 594 | ↗ FabrikamFi... | Minor UI updates |

Show all file changes

Francis Totten
Available - Video Capable
PROGRAM MANAGER

[✉](#) [✉](#) [📅](#) [✉](#) [✉](#)

Jamal Hartnett 5/5/2015 1
Francis Totten 5/1/2015 6
Jamal Hartnett 4/30/2015

3 references | 2/2 passing | Francis Totten, 3 hours ago [2 authors, 16 changes](#) 1 incoming change | 3 bugs | 4 work items

```
public ActionResult Create(Customer customer)
{
    if (customer == null)
```

Open the shortcut menu for an item to see the contact options. If you have Lync or Skype for Business installed, you see these options:

| ID | Branch | Description |
|-----|-----------------|--|
| 600 | ↗ FabrikamFi... | Updated UI strings for new season |
| 599 | ↗ FabrikamFi... | Fixed special character error in Create method |
| 598 | ↗ FabrikamFi... | Minor fix to code formatting |
| 596 | ↗ FabrikamFi... | Added guard helper class |
| 595 | ↗ FabrikamFi... | Minor UI updates |
| 594 | ↗ FabrikamFi... | Minor fix to formatting |

Show all file changes

3 references | 2/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change

Local Version

- [Compare with Local Version](#)
- [Changeset Details](#)
- [Track Changeset 600](#)
- [Hide Intermediate Branch Merges](#)
- [Send IM to Francis Totten](#)
- [Call Francis Totten](#)
- [Video Chat with Francis Totten](#)
- [Send Email to Francis Totten](#)
- [Open Contact Card](#)

Find unit tests for your code

Find out more about unit tests that exist for your code without opening Test Explorer. You'll need:

- Visual Studio Enterprise or Visual Studio Professional
- C# or Visual Basic code

- A [unit test project](#) that has unit tests for your application code

1. Go to application code that has unit tests.
2. Review the tests for that code (**Alt + 3**).

Test Duration

| Test | Duration |
|--|----------|
| ✖ CreateInsertsCustomerAndSaves | 854 ms |
| ✓ CreateNullCustomer | 11 ms |

Run All | Run

3 references | 0/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items

```
public ActionResult Create(Customer customer)
{
    if (customer == null)
```

3. If you see a warning icon , run the tests.

Test Duration

| Test | Duration |
|---|----------|
| ⚠ CreateInsertsCustomerAndSaves | |
| ⚠ CreateNullCustomer | |

Run All | Run

3 references | 0/2 passing | Francis Totten, 3 hours ago | 2 authors, 16 changes | 1 incoming change | 3 bugs | 4 work items

```
public ActionResult Create(Customer customer)
{
    if (customer == null)
```

4. To review a test's definition, double-click the test item in the CodeLens indicator window to open the code file in the editor.

```
[TestClass()]
0 references | Jia Hao Tseng | 1 author, 1 change | 1 work item
public class CustomersControllerTest
{
    MockCustomerRepository mockCustomerRepo;
    CustomersController controller;

    [TestInitialize()]
    0 references | Jia Hao Tseng | 1 author, 1 change | 1 work item
    public void SetupController()
    {
        mockCustomerRepo = new MockCustomerRepository();
        controller = new CustomersController(mockCustomerRepo);
    }

    [TestMethod()]
    ✖ 0 references | Jia Hao Tseng | 1 author, 1 change | 1 work item
    public void CreateInsertsCustomerAndSaves()
    {
        controller.Create(new Customer());

        Assert.IsTrue(mockCustomerRepo.IsInsertOrUpdateCalled);
        Assert.IsTrue(mockCustomerRepo.IsSaveCalled);
    }

    [TestMethod()]
    [ExpectedException(typeof(ArgumentNullException))]
    ✓ 0 references | Jia Hao Tseng | 1 author, 1 change | 1 work item
    public void CreateNullCustomer()
    {
        controller.Create(null);
    }
}
```

5. Review the test's results. Choose the test status indicator (✖ or ✓), or press **Alt + 1**.

✖ Test Failed - CreateInsertsCustomerAndSaves

Message: `Assert.IsTrue` failed.

Elapsed time: 854 ms

StackTrace:

```
CustomersControllerTest.CreateInsertsCustomerAndSaves()
```

Run | Debug

```
    [TestMethod()]
    public void CreateInsertsCustomerAndSaves()
    {
        controller.Create(new Customer());

        Assert.IsTrue(mockCustomerRepo.IsInsertOrUpdateCalled);
        Assert.IsTrue(mockCustomerRepo.IsSaveCalled);
    }
}
```

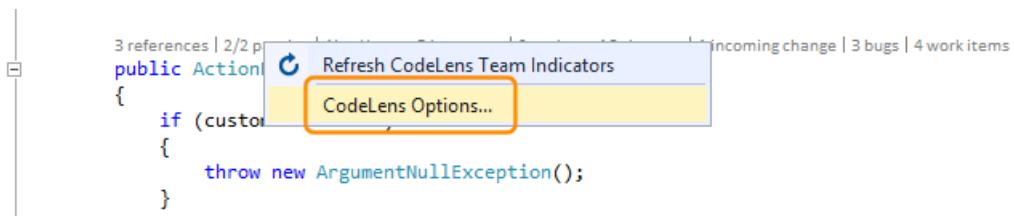
6. To see how many people changed this test, who changed this test, or how many changes were made to this test, [Find your code's history and linked items](#).

Q & A

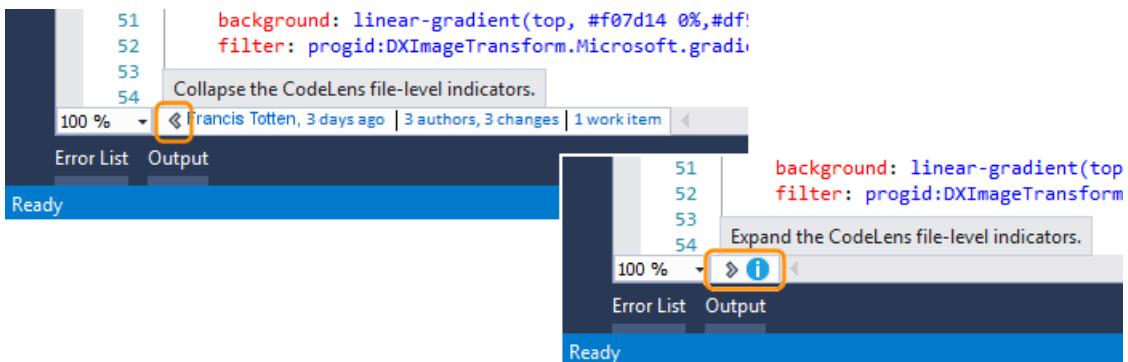
Q: How do I turn CodeLens off or on? Or choose which indicators to see?

A: You can turn indicators off or on, except for the references indicator. Go to **Tools, Options, Text Editor, All Languages, CodeLens**.

When the indicators are turned on, you can also open the CodeLens options from the indicators.



Turn CodeLens file-level indicators on and off using the chevron icons at the bottom of the editor window.



Q: Where is CodeLens?

A: CodeLens appears in C# and Visual Basic code at the method, class, indexer, and property level. CodeLens appears at the file level for all other types of files.

- Make sure CodeLens is turned on. Go to **Tools, Options, Text Editor, All Languages, CodeLens**.
- If your code is stored in TFS, make sure that code indexing is turned on by using the **CodeIndex command** with the [TFS Config command](#).
- TFS-related indicators appear only when work items are linked to the code and when you have permissions to open linked work items. [Confirm that you have team member permissions](#).
- Unit test indicators don't appear when application code doesn't have unit tests. Test status indicators appear

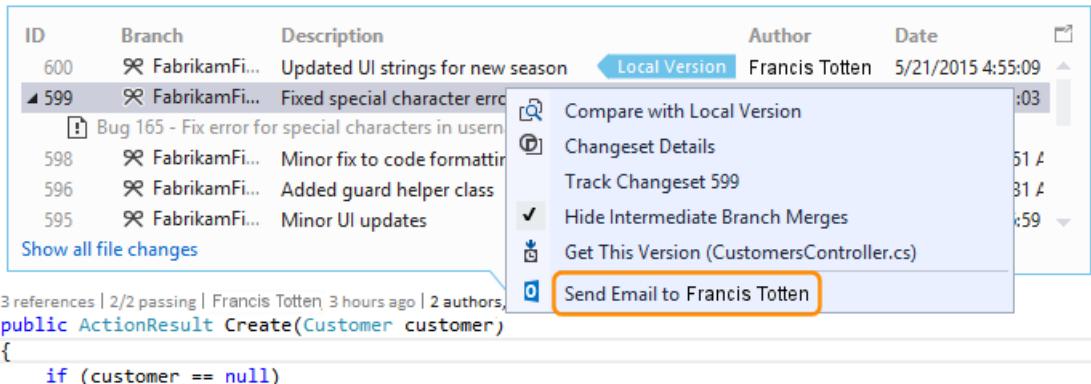
automatically in test projects. If you know that your application code has unit tests, but the test indicators don't appear, try building the solution (**Ctrl + Shift + B**).

Q: Why don't I see the work item details for a commit?

A: This might happen because CodeLens can't find the work items in TFS. Check that you're connected to the team project that has those work items and that you have permissions to see those work items. This might also happen if the commit description has incorrect information about the work item IDs in TFS.

Q: Why don't I see the Lync or Skype indicators?

A: They don't appear if you're not signed into Lync or Skype for Business, don't have one of these installed, or don't have a supported configuration. But you can still send mail:



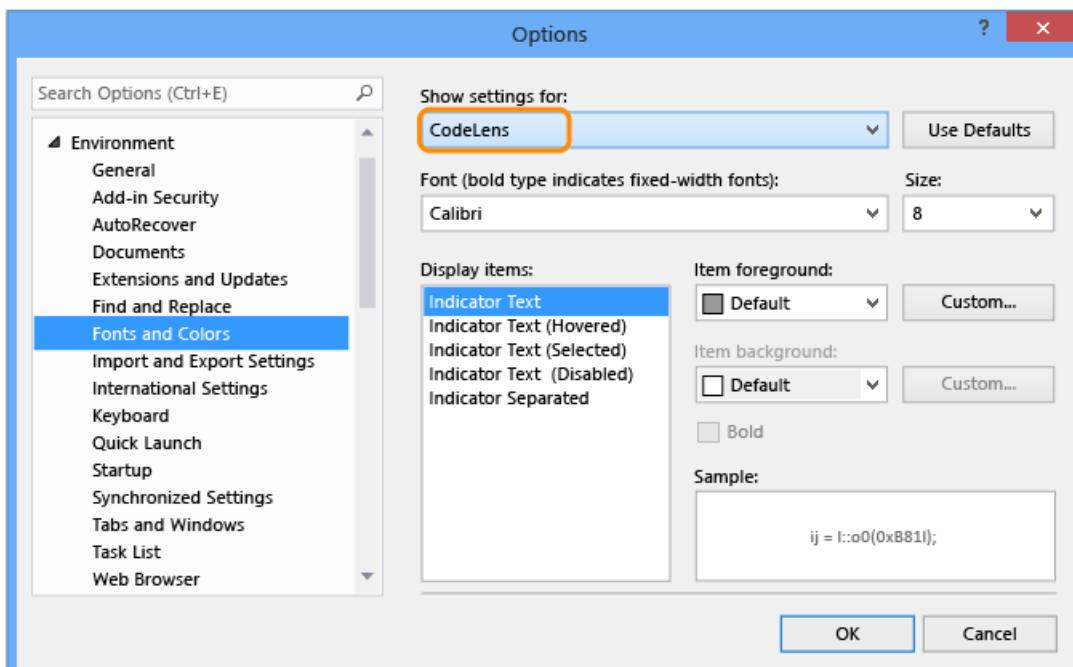
Which Lync and Skype configurations are supported?

- Skype for Business (32-bit or 64-bit)
- Lync 2010 or later alone (32-bit or 64-bit), but not Lync Basic 2013 with Windows 8.1

CodeLens doesn't support having different versions of Lync or Skype installed. They might not be localized for all localized versions of Visual Studio.

Q: How do I change the font and color for CodeLens?

A: Go to **Tools, Options, Environment, Fonts and Colors**.



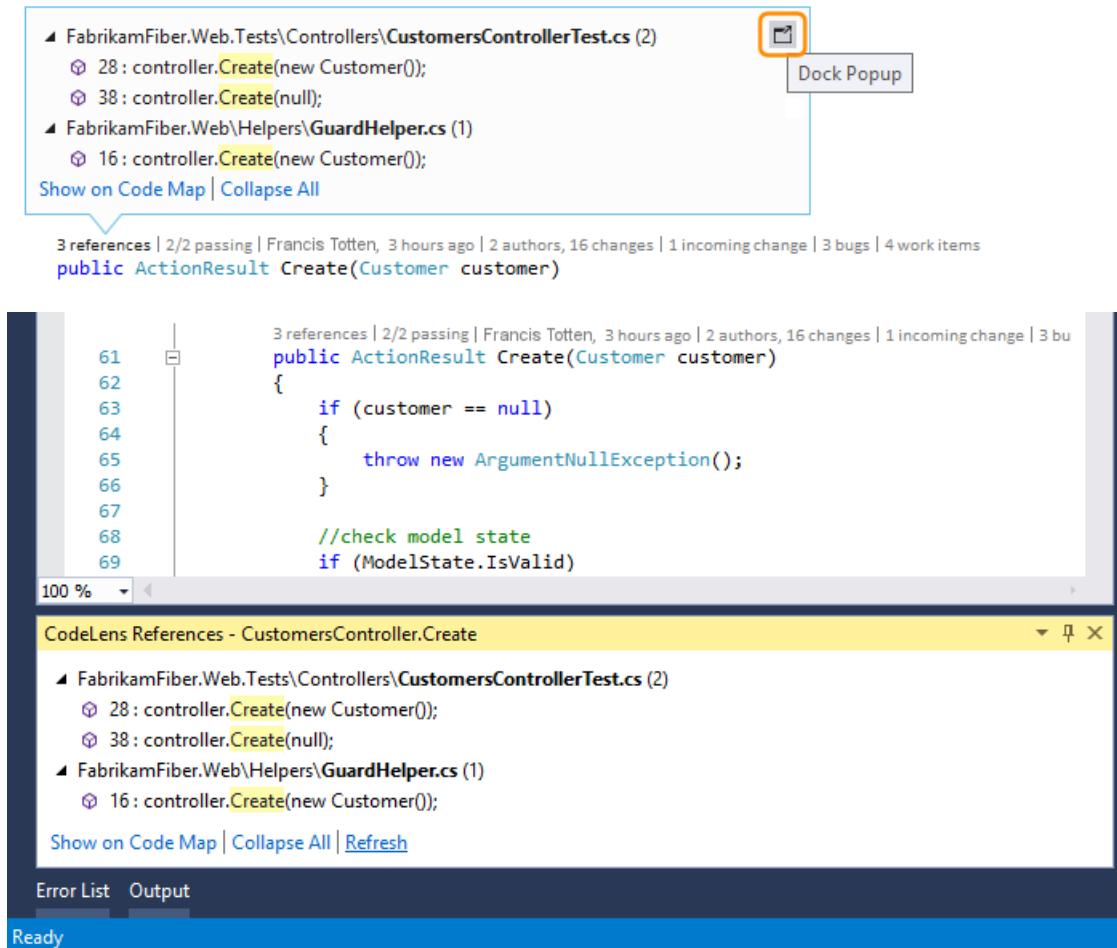
To use the keyboard:

1. Press **Alt + T + O** to open the **Options** box.

2. Press **Up Arrow** or **Down Arrow** to go to the **Environment** node, then press **Left Arrow** to expand the node.
3. Press **Down Arrow** to go to **Fonts and Colors**.
4. Press **TAB** to go to the **Show settings for** list, and then press **Down Arrow** to select **CodeLens**.

Q: Can I move the CodeLens heads-up display?

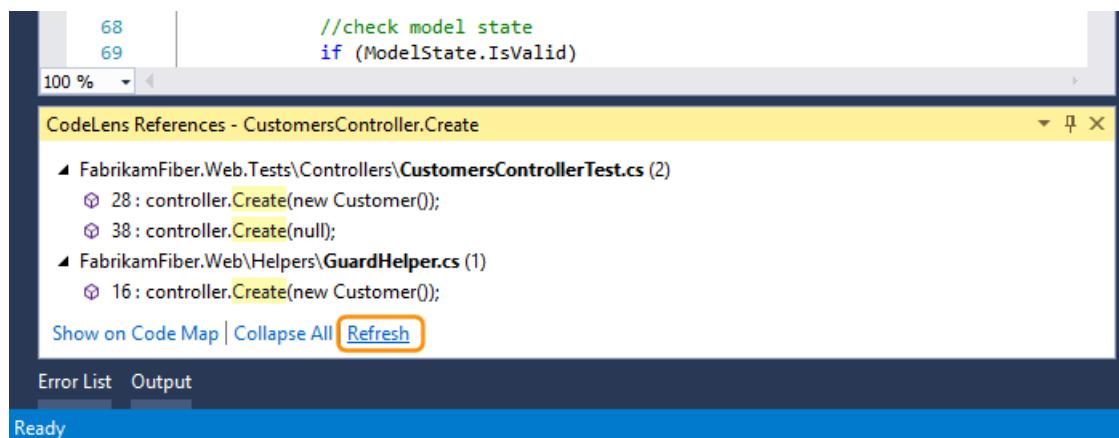
A: Yes, choose to dock CodeLens as a window.



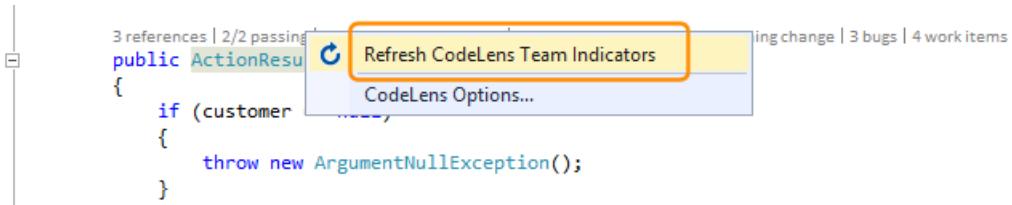
Q: How do I refresh the indicators?

A: This depends on the indicator:

- **References:** This indicator updates automatically when the code changes. If you have this indicator docked as a separate window, refresh the indicator manually here:



- **Team:** Refresh these indicators manually here:



- **Test:** Find unit tests for your code to refresh this indicator.

Q: What's "Local Version"?

A: The **Local Version** arrow points at the most recent changeset in your local version of this file. When the server has more recent changesets, they appear above or below the **Local Version** arrow, depending on the order used to sort the changesets.

Q: Can I manage how CodeLens processes code to show history and linked items?

A: Yes, if your code is in TFS, use the [CodeIndex command](#) with the [TFS Config command](#).

See also

[Writing Code in the Editor](#)

CodeIndex Command

1/26/2018 • 4 min to read • [Edit Online](#)

Use the **CodeIndex** command to manage code indexing on Team Foundation Server. For example, you might want to reset the index to fix CodeLens information, or turn off indexing to investigate server performance issues.

Required Permissions

To use the **CodeIndex** command, you must be a member of the **Team Foundation Administrators** security group. See [Permissions and groups defined for Team Services and TFS](#).

NOTE

Even if you log on with administrative credentials, you must open an elevated Command Prompt window to run this command. You must also run this command from the application tier for Team Foundation.

Syntax

```
TFSCmd CodeIndex /indexingStatus | /setIndexing:[ on | off | keepupOnly ] | /ignoreList:[ add | remove |  
removeAll | view ] ServerPath | /listLargeFiles [/fileCount:FileCount] [/minSize:MinSize] | /reindexAll |  
/destroyCodeIndex [/noPrompt] | /temporaryDataSizeLimit:[ view | <SizeInGBs> | disable ] |  
/indexHistoryPeriod:[ view | all | <NumberOfMonths> ] [/collectionName:CollectionName |  
/collectionId:CollectionId]
```

Parameters

| ARGUMENT | DESCRIPTION |
|-----------------------------|---|
| <code>CollectionName</code> | Specifies the name of the team project collection. If the name has spaces, enclose the name with quotation marks, for example, "Fabrikam Web Site". |
| <code>CollectionId</code> | Specifies the identification number of the team project collection. |
| <code>ServerPath</code> | Specifies the path to a code file. |

| OPTION | DESCRIPTION |
|---|---|
| <code>/indexingStatus</code> | Show the status and configuration of the code indexing service. |
| <code>/setIndexing:[on off keepupOnly]</code> | - on : Start indexing all changesets. - off : Stop indexing all changesets. - keepupOnly : Stop indexing previously created changesets and start indexing new changesets only. |

| OPTION | DESCRIPTION |
|--|---|
| /ignoreList: [add remove removeAll view] <input type="text" value="ServerPath"/> <p>You can use the wildcard character (*) at the start, end, or both ends of the server path.</p> | <p>Specifies a list of code files and their paths that you don't want indexed.</p> <ul style="list-style-type: none"> - add: Add the file that you don't want indexed to the ignored file list. - remove: Remove the file that you want indexed from the ignored file list. - removeAll: Clear the ignored file list and start indexing all files. - view: See all the files that aren't being indexed. |
| /listLargeFiles [/fileCount: <input <="" <input="" []=""]="" minsize:="" td="" type="text" value="MinSize"/> <td data-bbox="779 518 1447 653"> Shows the specified number of files that exceeds the specified size in KB. You can then use the /ignoreList option to exclude these files from indexing. </td> | Shows the specified number of files that exceeds the specified size in KB. You can then use the /ignoreList option to exclude these files from indexing. |
| /reindexAll | Clear previously indexed data and restart indexing. |
| /destroyCodeIndex [/noPrompt] | Delete the code index and remove all indexed data. Does not require confirmation if you use the /noPrompt option. |
| /temporaryDataSizeLimit: [view < <input type="text" value="SizeInGBs"/> > disable] | <p>Control how much temporary data that CodeLens creates when processing changesets. The default limit is 2 GB.</p> <ul style="list-style-type: none"> - view: Show the current size limit. - <input type="text" value="SizeInGBs"/> : Change the size limit. - disable: Remove the size limit. <p>This limit is checked before CodeLens processes a new changeset. If temporary data exceeds this limit, CodeLens will pause processing past changesets, not new ones. CodeLens will restart processing after the data is cleaned up and falls below this limit. Cleanup runs automatically once a day. This means temporary data might exceed this limit until cleanup starts running.</p> |
| /indexHistoryPeriod: [view all < <input type="text" value="NumberOfMonths"/> >] | <p>Control how long to index your change history. This affects how much history CodeLens shows you. The default limit is 12 months. This means CodeLens shows your change history from the last 12 months only.</p> <ul style="list-style-type: none"> - view: Show the current number of months. - all: Index all change history. - <input type="text" value="NumberOfMonths"/> : Change the number of months used to index change history. |
| /collectionName: <input type="text" value="CollectionName"/> | Specifies the name of the team project collection on which to run the CodeIndex command. Required if you don't use /CollectionId . |
| /collectionId: <input type="text" value="CollectionId"/> | Specifies the identification number of the team project collection on which to run the CodeIndex command. Required if you don't use /CollectionName . |

Examples

NOTE

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be inferred.

To see the code indexing status and configuration:

```
TFSCfg Config CodeIndex /indexingStatus /collectionName:"Fabrikam Web Site"
```

To start indexing all changesets:

```
TFSCfg Config CodeIndex /setIndexing:on /collectionName:"Fabrikam Web Site"
```

To stop indexing previously created changesets and start indexing new changesets only:

```
TFSCfg Config CodeIndex /setIndexing:keepupOnly /collectionName:"Fabrikam Web Site"
```

To find up to 50 files that are larger than 10 KB:

```
TFSCfg Config CodeIndex /listLargeFiles /fileCount:50 /minSize:10 /collectionName:"Fabrikam Web Site"
```

To exclude a specific file from indexing and add it to the ignored file list:

```
TFSCfg Config CodeIndex /ignoreList:add "$/Fabrikam Web Site/Catalog.cs" /collectionName:"Fabrikam Web Site"
```

To see all the files that aren't indexed:

```
TFSCfg Config CodeIndex /ignoreList:view
```

To clear previously indexed data and restart indexing:

```
TFSCfg Config CodeIndex /reindexAll /collectionName:"Fabrikam Web Site"
```

To save all changeset history:

```
TFSCfg Config CodeIndex /indexHistoryPeriod:all /collectionName:"Fabrikam Web Site"
```

To remove the size limit on CodeLens temporary data and continue indexing regardless of temporary data size:

```
TFSCfg Config CodeIndex /temporaryDataSizeLimit:disable /collectionName:"Fabrikam Web Site"
```

To delete the code index with confirmation:

```
TFSCfg Config CodeIndex /destroyCodeIndex /collectionName:"Fabrikam Web Site"
```

See also

[Find code changes and other history with CodeLens](#)

[Managing server configuration with TFSConfig](#)

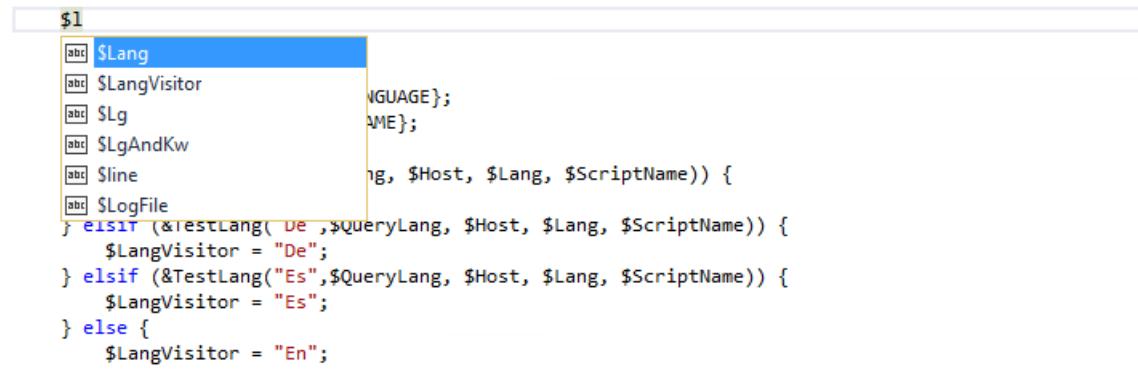
Adding Visual Studio editor support for other languages

1/26/2018 • 2 min to read • [Edit Online](#)

Learn about how the Visual Studio editor supports reading and navigating through different computer languages and how you can add Visual Studio editor support for other languages.

Syntax colorization, statement completion, and Navigate To support

Features in the Visual Studio editor such as syntax colorization, statement completion, and Navigate To can help you more easily read, create, and edit your code. The following screenshot shows an example of editing a Perl script in Visual Studio. The syntax is automatically colorized. For example, remarks in the code are colored green, code is black, paths are red, and statements are blue. The Visual Studio editor automatically applies syntax colorization to any language it supports. In addition, when you begin to enter a known language keyword or object, statement completion displays a list of possible statements and objects. Statement completion can help you create code more quickly and easily.



```
81      if ($ENV{'HTTP_USER_AGENT'} =~ /scooter|robot|ultraseek|spider|wget/i) {
82          return;
83          # En: Don't Display Adverts for robots.
84          # Fr: N'affiche pas la pub pour les robots.
85      }
86
87
88      if ($ENV{'HTTP_USER_AGENT'} =~ /Win/i) { $OSVisitor = "1" }
89      elsif ($ENV{'HTTP_USER_AGENT'} =~ /Macintosh/i) { $OSVisitor = "2" }
90      elsif ($ENV{'HTTP_USER_AGENT'} =~ /Linux/i) { $OSVisitor = "3" }
91      elsif ($ENV{'HTTP_USER_AGENT'} =~ /X11/i) { $OSVisitor = "4" }
92      elsif ($ENV{'HTTP_USER_AGENT'} =~ /scooter|robot|ultraseek|spider|wget/i) { $OSVisitor = "5" }
93      else { $OSVisitor = "6" }
94
95      $1
96      $Lang
97      $LangVisitor
98      $Lg
99      $LgAndKw
100     $Line
101     $LogFile
102
103 } elsif (&TestLang("de", $QueryLang, $Host, $Lang, $ScriptName)) {
104     $LangVisitor = "De";
105 } elsif (&TestLang("es", $QueryLang, $Host, $Lang, $ScriptName)) {
106     $LangVisitor = "Es";
107 } else {
108     $LangVisitor = "En";
109 }
```

Visual Studio currently provides syntax colorization and basic statement completion support for the following languages using [TextMate Grammars](#). If your favorite language isn't in the table, though, don't worry - you can add it.

| Bat | F# | Java | Markdown | Rust | Visual Basic |
|---------|--------|---------|-------------|-------------|--------------|
| Clojure | Go | JavaDoc | Objective-C | ShaderLab | C# |
| CMake | Groovy | JSON | Perl | ShellScript | Visual C++ |

| | | | | | |
|--------------|------|------|--------|------------|-------|
| CoffeeScript | HTML | LESS | Python | SQL | VBNet |
| CSS | INI | LUA | R | Swift | XML |
| Docker | Jade | Make | Ruby | TypeScript | YAML |

In addition to syntax colorization and basic statement completion, Visual Studio also has a feature called [Navigate To](#). This feature enables you to quickly search code files, file paths and code symbols. Visual Studio provides Navigate To support for the following languages.

- Go
- Java
- JavaScript
- PHP
- TypeScript
- Visual Basic
- Visual C++
- C#

All of these file types have the features described earlier even if support for a given language hasn't yet been installed. Installing specialized support for some languages may provide additional language support, such as IntelliSense or other advanced language features such as Light Bulbs.

Adding support for non-supported languages

Visual Studio 2015 Update 1 and later versions provide language support in the editor by using [TextMate Grammars](#). If your favorite programming language currently isn't supported in the Visual Studio editor, first, search the web - a TextMate bundle for the language may already exist. If you can't find one, though, you can add support for it yourself in Visual Studio 2015 Update 1 or later by creating a TextMate bundle model for language grammars and snippets.

Add any new TextMate Grammars for Visual Studio in the following folder:

```
%userprofile%\vs\Extensions
```

Under this base path, add the following folder(s) if they apply to your situation:

| FOLDER NAME | DESCRIPTION |
|------------------|---|
| \<language name> | The language folder. Replace <language name> with the name of the language. For example, \Matlab. |
| \Syntaxes | The grammar folder. Contains the grammar .json files for the language, such as Matlab.json . |
| \Snippets | The snippets folder. Contains snippets for the language. |

In Windows, %userprofile% resolves to the path: c:\Users\<user name>. If the extensions folder does not exist on your system, you will need to create it. If the folder already exists, it will be hidden.

For details about how to create TextMate Grammars, see [TextMate - Introduction to Language Grammars: How to add source code syntax highlighting embedded in HTML](#) and [Notes on how to create a Language Grammar and Custom Theme for a Textmate Bundle](#).

See Also

- [Visual Studio 2013 Navigate To Improvements](#)
- [Walkthrough: Creating a Code Snippet](#)
- [Walkthrough: Displaying Statement Completion](#)

Cross-Platform mobile development in Visual Studio

3/21/2018 • 10 min to read • [Edit Online](#)

You can build apps for Android, iOS, and Windows devices by using Visual Studio. As you design your app, use tools in Visual Studio to easily add connected services such as Office 365, Azure App Service, and Application Insights.

Build your apps by using C# and the .NET Framework, HTML and JavaScript, or C++. Share code, strings, images, and in some cases even the user interface.

If you want to build a game or immersive graphical app, install Visual Studio tools for Unity and enjoy all of the powerful productivity features of Visual Studio with Unity, the popular cross-platform game/graphics engine and development environment for apps that run on iOS, Android, Windows, and other platforms.

Build an app for Android, iOS, and Windows (.NET Framework)



With Xamarin, you can target Android, iOS, and Windows in the same solution, sharing code and even UI.

LEARN MORE

[Install Visual Studio](#) (VisualStudio.com)

[Learn about Xamarin in Visual Studio](#) (VisualStudio.com)

[Visual Studio and Xamarin](#) (MSDN Library)

[Application Lifecycle Management \(ALM\) with Xamarin apps](#) (MSDN Library)

[Learn about Universal Windows apps in Visual Studio](#) (VisualStudio.com)

[Learn about the similarities between Swift and C#](#) (download.microsoft.com)

[Learn about the Visual Studio Emulator for Android](#) (VisualStudio.com)

Target Android, iOS, and Windows from a single code base

You can build native apps for Android, iOS, and Windows by using C# or F# (Visual Basic is not supported at this time). To get started, install Visual Studio 2015, select the **Custom** option in the installer, and check the box under **Cross Platform Mobile Development > C#/NET (Xamarin)**. You can also start with the [Xamarin Installer](#), which is required to install Xamarin for Visual Studio 2013.

If you already have Visual Studio 2015 installed, run the installer from **Control Panel > Programs and Features** and select the same **Custom** option for Xamarin as above.

When you're done, project templates appear in the **New Project** dialog box. The easiest way to find Xamarin templates is to just search on "Xamarin."

Xamarin exposes the native functionality of Android, iOS, and Windows as .NET objects. Thus your apps have full access to native APIs and native user controls, and they're just as responsive as apps written in the native platform languages.

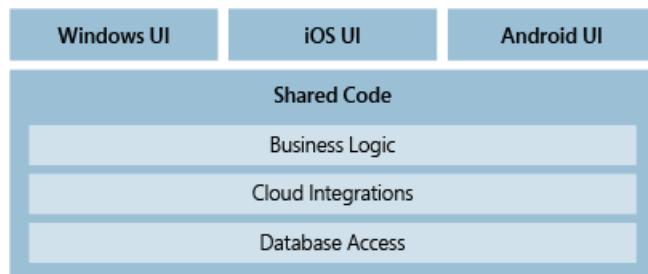
After you create a project, you'll leverage all of the productivity features of Visual Studio. For example, you'll use a designer to create your pages, and use IntelliSense to explore the native API's of the mobile platforms. When you're ready to run your app and see how it looks, you can use the Visual Studio Emulator for Android or the Android SDK emulator, run Windows apps natively, or run Windows apps on the Windows Phone emulator. You can also use tethered Android and Windows devices directly. For iOS projects, connect to a networked Mac and start the Mac emulator from Visual Studio, or connect to a tethered device.

Design one set of pages that render across all devices by using [Xamarin.Forms](#)

Depending on the complexity of your apps design, you might consider building it by using [Xamarin.Forms](#) templates in the **Mobile Apps** group of project templates. Xamarin.Forms is a UI toolkit that lets you create a single interface that you can share across Android, iOS, and Windows. When you compile a Xamarin.Forms solution, you'll get an Android app, an iOS app, and a Windows app. For more details, see [Learn about mobile development with Xamarin](#).

Share code between Android, iOS, and Windows apps

If you're not using Xamarin.Forms and choose to design for each platform individually, you can share most of your non-UI code between platform projects (Android, iOS, and Windows). This includes any business logic, cloud integration, database access, or any other code that targets the .NET Framework. The only code that you can't share is code that targets a specific platform.



You can share your code by using a shared project, a Portable Class Library project, or both. You might find that some code fits best in a shared project, and some code makes more sense inside a Portable Class Library project.

LEARN MORE

Choose whether to share your code by using shared projects, Portable Class Library projects, or both.

[Sharing code across platforms \(.NET Framework blog\)](#)

[Sharing Code Options \(Xamarin\)](#)

[Code sharing options with the .NET Framework \(MSDN Library\)](#)

Target Windows 10 devices



If you want to create a single app that targets the full breadth of Windows 10 devices, create a universal Windows app. You'll design the app by using a single project and your pages will render properly no matter what device is used to view them.

Start with a universal Windows app project template. Design your pages visually, and then open them in a preview

window to see how they appear for various types of devices. If you don't like how a page appears on a device, you can optimize the page to better fit the screen size, resolution, or various orientations such as landscape or portrait mode. You can do all of that by using intuitive tool windows and easily accessible menu options in Visual Studio. When you're ready to run your app and step through your code, you'll find all of the device emulators and simulators for different types of devices together in one drop-down list that is located on the **Standard** toolbar.

Windows 10 is fairly new, so you'll also find project templates that target Windows 8.1. You can use those project templates if you want and your app will run on Windows 10 phones, tablets, and PCs. However, all devices that run Windows 8.1 will receive an automatic upgrade to Windows 10, so unless you have specific reasons why you'd rather target Windows 8.1, we recommend that you use the project templates that target Windows 10.

LEARN MORE

[Intro to the Universal Windows Platform](#)

[Create your first app](#)

[Develop apps for the Universal Windows Platform \(UWP\)](#)

[Migrate apps to the Universal Windows Platform \(UWP\)](#)

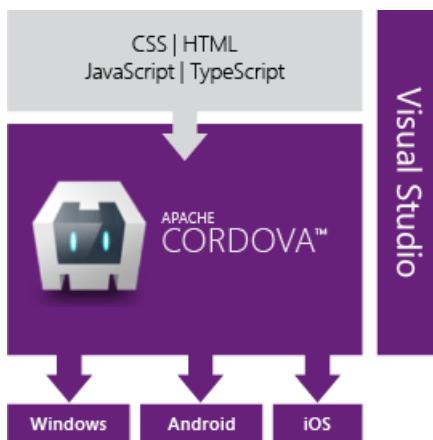
Build an app for Android, iOS, and Windows (HTML/JavaScript)



If you're a web developer, and you're familiar with HTML and JavaScript, you can target Windows, Android, and iOS by using Visual Studio Tools for Apache Cordova. These apps can target all three platforms and you can build them by using the skills and processes that you're most familiar with.

Apache Cordova is a framework that includes a plug-in model. This plug-in model provides a single JavaScript API that you can use to access the native device capabilities of all three platforms (Android, iOS, and Windows).

Because these APIs are cross-platform, you can share most of what you write between all three platforms. This reduces your development and maintenance costs. Also, there's no need to start from scratch. If you've created other types of web applications, you can share those files with your Cordova app without having to modify or redesign them in any way.



To get started, install Visual Studio 2015 and choose the **HTML/JavaScript (Apache Cordova)** feature during setup. If you're using Visual Studio 2013, install the Visual Studio Tools for Apache Cordova extension. Either way, the Cordova tools automatically install all third-party software that's required to build your multi-platform app.

After you've installed the extension, open Visual Studio and create a **Blank App (Apache Cordova)** project. Then,

you can develop your app by using JavaScript or Typescript. You can also add plug-ins to extend the functionality of your app, and APIs from plug-ins appear in IntelliSense as you write code.

When you're ready to run your app and step through your code, choose an emulator, such as the Apache Ripple emulator or Visual Studio Emulator (Android or Windows Phone), a browser, or a device that you've connected directly to your computer. Then, start your app. If you're developing your app on a Windows PC, you can even run it on that. All of these options are built into Visual Studio as part of the Visual Studio Tools for Apache Cordova.

Project templates for creating universal Windows apps are still available in Visual Studio so feel free to use them if you plan to target only Windows devices. If you decide to target Android and iOS later, you can always port your code to a Cordova project. There are open-source versions of the WinJS APIs, so you can reuse any code that consumes those APIs. That said, if you plan to target other platforms in the future, we recommend that you start with the Visual Studio Tools for Apache Cordova.

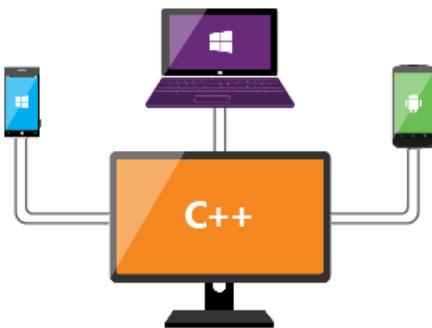
LEARN MORE

[Install Visual Studio](#) (VisualStudio.com)

[Get started with Visual Studio Tools for Apache Cordova](#) (docs.microsoft.com)

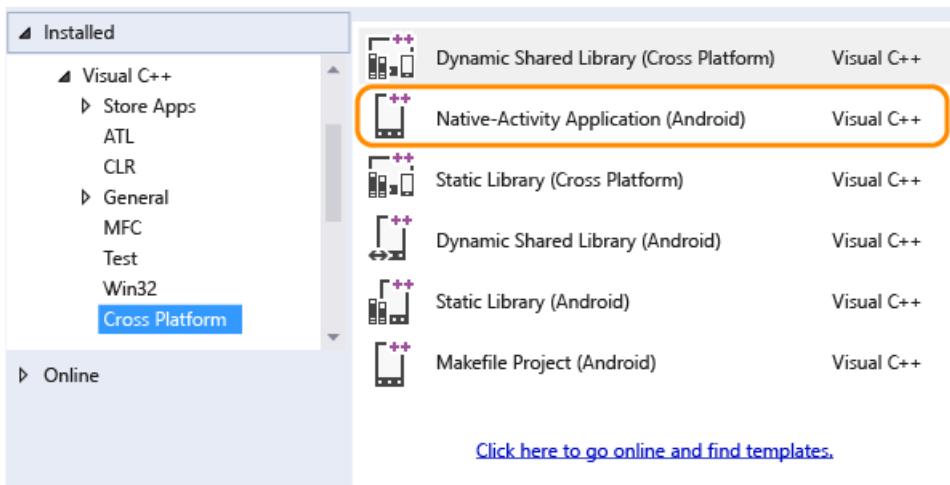
[Learn about the Visual Studio Emulator for Android](#) (VisualStudio.com)

Build an app for Android and Windows (C++)



First, install Visual Studio 2015 and the Visual C++ for Cross Platform Mobile Development tools. Then, you can build a native activity application for Android or an app that targets Windows. C++ templates that target iOS are not yet available. You can target Android and Windows in the same solution if you want, and then share code between them by using a cross-platform static or dynamic shared library.

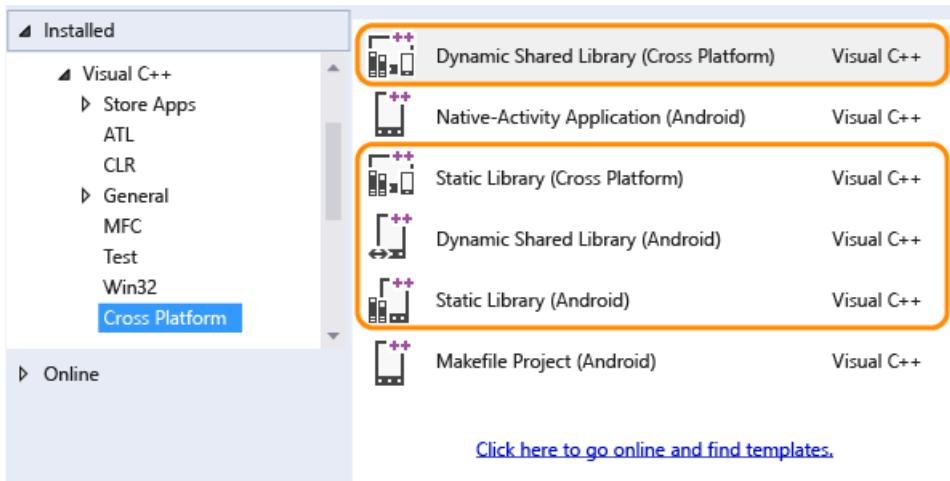
If you need to build an app for Android that requires any sort of advanced graphics manipulation, such as a game, you can use C++ to do it. Start with the **Native-Activity Application (Android)** project. This project has full support for the Clang toolchain.



When you're ready to run your app and see how it looks, use the Visual Studio Emulator for Android. It's fast, reliable, and easy to install and configure.

You can also build an app that targets the full breadth of Windows 10 devices by using C++ and a universal Windows app project template. Read more about this in the [Target Windows 10 devices](#) section that appears earlier in this topic.

You can share C++ code between Android and Windows by creating a static or dynamic shared library.



You can consume that library in a Windows or Android project, like the ones described earlier in this section. You can also consume it in an app that you build by using Xamarin, Java, or any language that lets you invoke functions in an unmanaged DLL.

As you write code in these libraries, you can use IntelliSense to explore the native APIs of the Android and Windows platforms. These library projects are fully integrated with the Visual Studio debugger so you can set breakpoints, step through code, and find and fix issues by using all of the advanced features of the debugger.

LEARN MORE

[Download Visual Studio.](#) ([VisualStudio.com](#))

[Install the Visual C++ for Cross-Platform Mobile Development tools.](#) ([MSDN Library](#))

[Learn more about using C++ to target multiple platforms.](#) ([VisualStudio.com](#))

[Install what you need, and then create a native activity application for Android](#) ([MSDN Library](#))

[Learn about the Visual Studio Emulator for Android](#) ([VisualStudio.com](#))

LEARN MORE

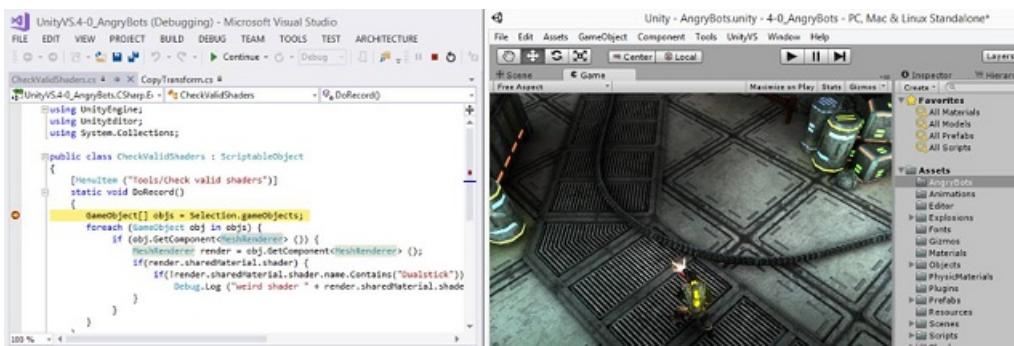
[Learn more about sharing C++ code with Android and Windows apps \(VisualStudio.com\)](#)

[Cross-platform mobile development examples for C++ \(MSDN Library\)](#)

[Additional cross-platform mobile development examples for C++ \(code.msdn\)](#)

Build a cross-platform game for Android, iOS, and Windows by using Visual Studio tools for Unity

Visual Studio Tools for Unity is a free extension for Visual Studio that integrates Visual Studio's powerful code editing, productivity, and debugging tools with *Unity*, the popular cross-platform gaming/graphics engine and development environment for immersive apps that target Windows, iOS, Android, and other platforms including the web.



With Visual Studio Tools for Unity (VSTU), you can use Visual Studio to write game and editor scripts in C# and then use its powerful debugger to find and fix errors. The latest release of VSTU brings support for Unity 5 and includes syntax coloring for Unity's ShaderLab shader language, better synchronization with Unity, richer debugging, and improved code generation for the MonoBehavior wizard. VSTU also brings your Unity project files, console messages, and the ability to start your game into Visual Studio so you can spend less time switching to and from the Unity Editor while writing code.

Start building your game with Unity and Visual Studio Tools for Unity today.

LEARN MORE

[Learn more about building Unity games with Visual Studio](#)

[Read more about Visual Studio Tools for Unity \(MSDN Library\)](#)

[Start using Visual Studio Tools for Unity \(MSDN Library\)](#)

[Read about the latest enhancements to the Visual Studio Tools for Unity 2.0 Preview \(Visual Studio blog\)](#)

[Watch a video introduction to the Visual Studio Tools for Unity 2.0 Preview \(Video\)](#)

[Learn about Unity \(Unity website\)](#)

See Also

- [Add Office 365 API's to a Visual Studio project](#)
- [Azure App Services - Mobile Apps](#)

- [HockeyApp for mobile](#)

Office and SharePoint Development in Visual Studio

1/10/2018 • 3 min to read • [Edit Online](#)

You can extend Microsoft Office and SharePoint by creating a lightweight app or add-in that users download from the [Office Store](#) or an organizational catalog, or by creating a .NET Framework-based solution that users install on a computer.

In this topic:

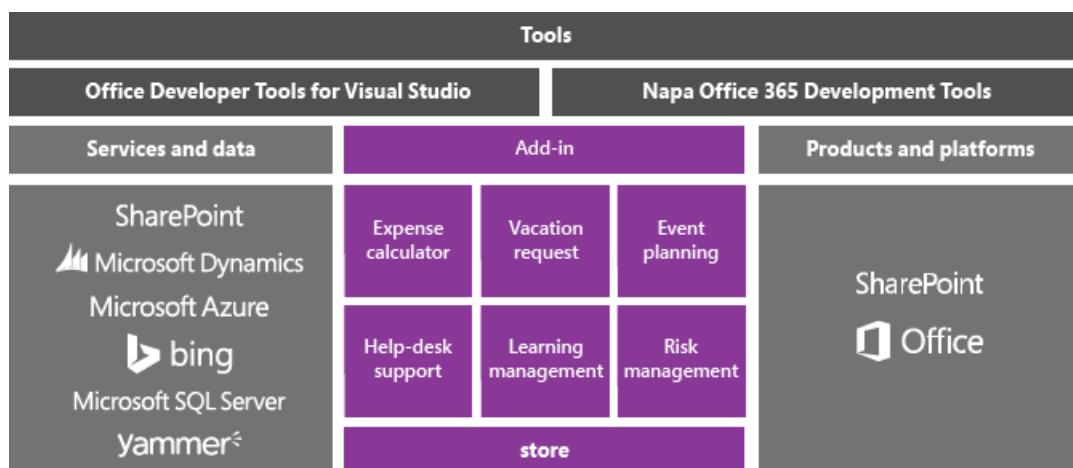
- [Create add-ins for Office and SharePoint](#)
- [Create a VSTO add-in](#)
- [Create a SharePoint solution](#)

Create add-ins for Office and SharePoint

Office 2013 and SharePoint 2013 introduce a new add-in model that helps you build, distribute, and monetize add-ins that extend Office and SharePoint. These add-ins can run in Office or SharePoint Online, and users can interact with them from many devices.

Find out how to use the new [Office add-in model](#) to extend the Office experience for your users.

These add-ins have very small footprints compared to VSTO add-ins and solutions, and you can build them by using almost any web programming technology such as HTML5, JavaScript, CSS3, and XML. To get started, use the Office Developer Tools in Visual Studio, or the lightweight web-based tool code-named Napa Office 365 Development Tools, which lets you create projects, write code, and run your add-ins in a browser.



Learn more

| TO | SEE |
|---|---|
| Learn more about Napa Office 365 Development Tools. | Napa Office 365 Development Tools |

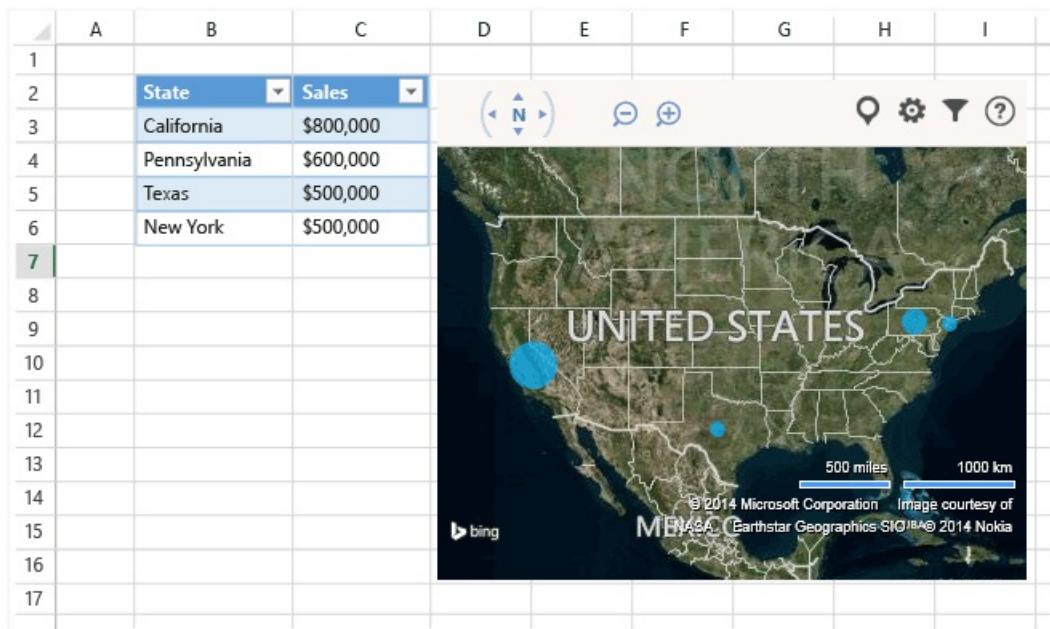
Build an Office add-in

To extend the functionality of Office, build an Office add-in. It's basically a webpage that's hosted in an Office application such as Excel, Word, Outlook, and PowerPoint. Your app can add functionality to documents, worksheets, email messages, appointments, presentations, and projects.

You can sell your app in the Office Store. The [Office Store](#) makes it easy to monetize your add-ins, manage updates,

and track telemetry. You can also publish your app to users through an app catalog in SharePoint, or on Exchange Server.

The following app for Office shows worksheet data in a Bing map.



Learn more

| TO | SEE |
|--|---|
| Learn more about Office add-ins, and then build one. | Office add-ins |
| Compare the different ways in which you can extend Office, and decide whether you should use an app or an Office add-in. | Roadmap for Office add-ins, VSTO, and VBA |
| Learn more about Napa Office 365 Development Tools. | Napa Office 365 Development Tools |

Build a SharePoint add-in

To extend SharePoint for your users, build a SharePoint add-in. It's basically a small, easy-to-use, stand-alone application that solves a need for your users or business.

You can sell your app for SharePoint in the [Office Store](#). You can also publish your add-in to users through an add-in catalog in SharePoint. Site owners can install, upgrade, and uninstall your add-in on their SharePoint sites without the help of a farm server or site collection administrator.

Here's an example of an app for SharePoint that helps users manage business contacts.

The screenshot shows the Business Contact Manager application. On the left, there's a navigation sidebar with icons for Contacts, Companies, Opportunities, Products, and About. The main area has tabs for 'List' and 'Datasheet'. A search bar says 'Filter the list...'. Below it are four contact entries:

| | First name | Company | Action |
|--|-----------------|-----------------|--------|
| | Nancy Anderson | Contoso, Ltd | |
| | Ryan McMinn | MSFT | |
| | Satomi Hayakawa | Adventure Works | |

On the right, there's a partial view of another section labeled 'Opportunit...'.

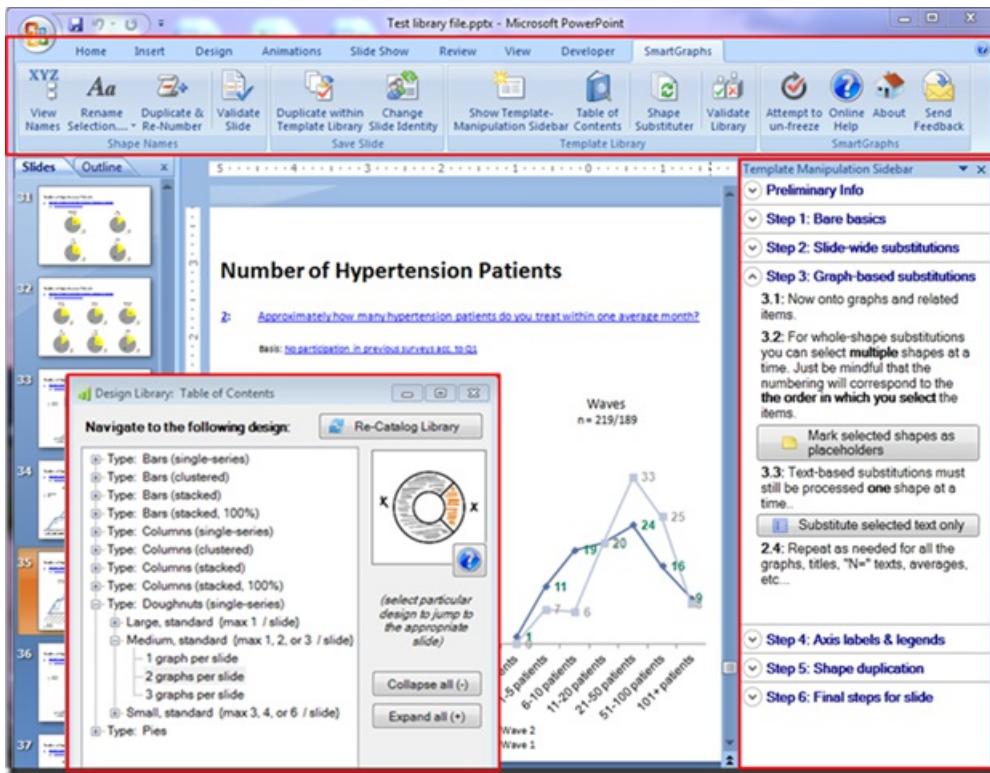
Learn more

| TO | SEE |
|---|--|
| Learn more about SharePoint add-ins, and then build one. | SharePoint Add-ins |
| Compare add-ins for SharePoint with traditional SharePoint solutions. | SharePoint add-ins compared with SharePoint solutions |
| Choose whether to build a SharePoint add-in or a SharePoint solution. | Deciding between SharePoint add-ins and SharePoint solutions |
| Learn more about Napa Office 365 Development Tools. | Napa Office 365 Development Tools |

Create a VSTO add-in

Create a VSTO add-in to target Office 2007 or Office 2010, or to extend Office 2013 and Office 2016 beyond what's possible with Office add-ins. VSTO add-ins run only on the desktop. Users have to install VSTO add-ins, so they're typically more difficult to deploy and support. However, your VSTO add-in can be integrated more closely with Office. For example, it can add tabs and controls to the Office Ribbon and perform advanced automation tasks such as merging documents or modifying charts. You can leverage the .NET Framework and use C# and Visual Basic to interact with Office objects.

Here's an example what a VSTO add-in can do. This VSTO add-in adds Ribbon controls, a custom task pane, and a dialog box to PowerPoint.



Learn more

| TO | READ |
|---|---|
| Compare the different ways in which you can extend Office, and decide whether you should use a VSTO add-in or an Office add-in. | Roadmap for Office add-ins, VSTO, and VBA |
| Create a VSTO add-in. | VSTO add-ins build with Visual Studio |

Create a SharePoint solution

Create a SharePoint solution to target SharePoint Foundation 2010 and SharePoint Server 2010, or to extend SharePoint 2013 and SharePoint 2016 in ways beyond what's possible with a SharePoint add-in.

SharePoint solutions require on-premises SharePoint farm servers. Administrators must install them, and because solutions execute in SharePoint, they can affect the performance of the server. However, solutions provide deeper access to SharePoint objects. Also, when you build a SharePoint solution, you can leverage the .NET Framework and use C# and Visual Basic to interact with SharePoint objects.

Learn more

| TO | SEE |
|---|---|
| Compare SharePoint solutions with SharePoint add-ins. | SharePoint add-ins compared with SharePoint solutions |
| Create a SharePoint solution. | Create SharePoint Solutions |

XML Tools in Visual Studio

1/25/2018 • 2 min to read • [Edit Online](#)

Extensible Markup Language (XML) is a markup language that provides a format for describing data. This facilitates more precise declarations of content and more meaningful search results across multiple platforms. In addition, XML enables the separation of presentation from data. For example, in HTML you use tags to tell the browser to display data as bold or italic; in XML you use tags only to describe data, such as city name, temperature, and barometric pressure. In XML you use style sheets such as Extensible Stylesheet Language (XSL) and cascading style sheets (CSS) to present the data in a browser. XML separates the data from the presentation and the process. This enables you to display and process the data as you want to, by applying different style sheets and applications.

XML is a subset of SGML that is optimized for delivery over the Web. It is defined by the World Wide Web Consortium (W3C). This standardization guarantees that structured data will be uniform and independent of applications or vendors.

XML is at the core of many features of Visual Studio and the .NET Framework. The following topic list names the tools and features related to XML that are offered in Visual Studio and the .NET Framework.

For more information, see the [System.Xml](#) documentation.

In This Section

[Working with XML Data](#)

Discusses the role of XML in the way data is handled in Visual Studio.

[Debugging XSLT](#)

Provides links to topics about using the Visual Studio debugger to debug XSLT.

Reference

[Microsoft.VisualStudio.XmlEditor](#)

Exposes the [XML Editor](#) parse tree through [System.Xml.Linq](#) for any XML documents.

[XML Standards Reference](#)

Provides information about XML technologies, including XML, Document Type Definition (DTD), XML Schema definition language (XSD), and XSLT.

[System.Xml](#)

Describes the classes and other elements that make up the [System.Xml](#) namespace and provides links to more detailed information on each item.

[System.Xml.Serialization](#)

Describes the classes and other elements that make up the [System.Xml.Serialization](#) namespace and provides links to more detailed information about each item.

Related Sections

[XML Document Object Model \(DOM\)](#)

Describes how the [XmlDocument](#) and its associated classes comply with the W3C Document Object Model (Core) Level 1 and Level 2 namespace support specifications.

[Processing XML Data with XmlReader and XmlWriter](#)

XSLT Transformations

Describes how the [XslCompiledTransform](#) class implements the XSLT 1.0 recommendation.

Process XML Data Using the XPath Data Model

Describes how the [XPathNavigator](#) class can process XML data stored in an [XPathDocument](#) or an [XmlDocument](#) object. The [XPathNavigator](#) class is based on the XQuery 1.0 and XPath 2.0 Data Model and can be used to navigate and edit XML data.

XML Schema Object Model (SOM)

Describes the classes used for creating and manipulating XML Schemas, by providing an [XmlSchema](#) class to load and edit a schema.

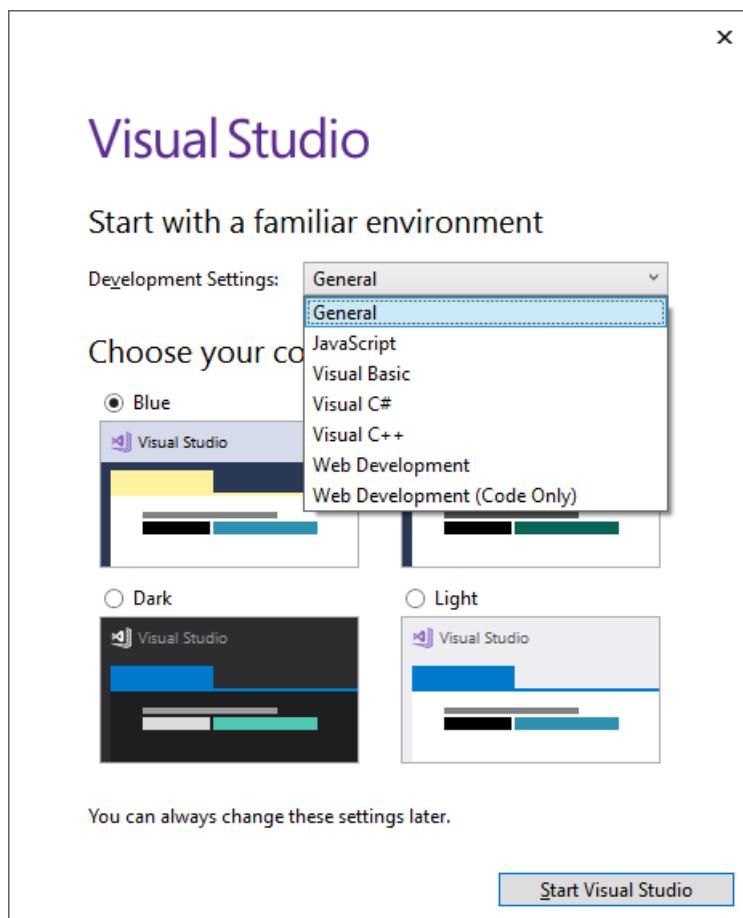
Walkthrough: Create a Simple Application with C# or Visual Basic

1/26/2018 • 8 min to read • [Edit Online](#)

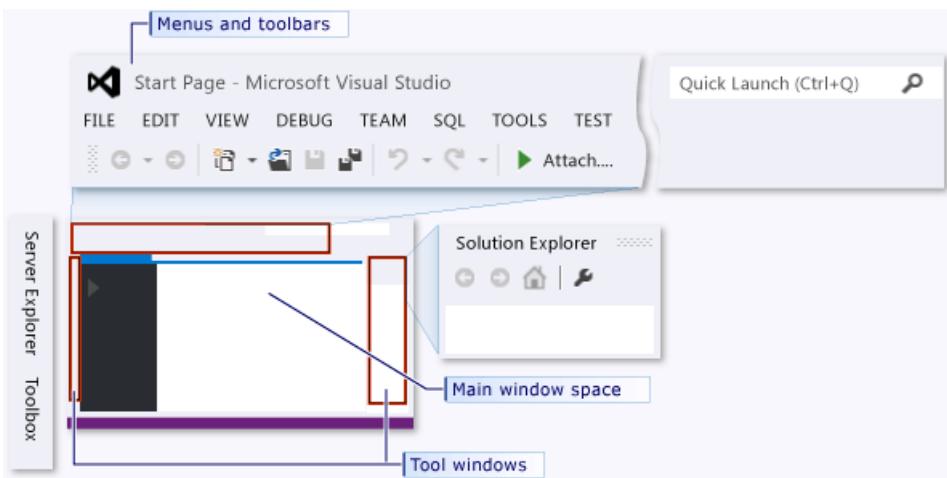
By completing this walkthrough, you'll become familiar with many of the tools, dialog boxes, and designers that you can use when you develop applications with Visual Studio. You'll create a simple "Hello, World" application, design the UI, add code, and debug errors, while you learn about working in the integrated development environment (IDE).

Configure the IDE

When you start Visual Studio for the first time, you'll be prompted to sign in. This step is optional for this walkthrough. Next you may be shown a dialog box that asks you to choose your development settings and color theme. Keep the defaults and choose **Start Visual Studio**.



After Visual Studio launches, you'll see tool windows, the menus and toolbars, and the main window space. Tool windows are docked on the left and right sides of the application window, with **Quick Launch**, the menu bar, and the standard toolbar at the top. In the center of the application window is the **Start Page**. When you load a solution or project, editors and designers appear in the space where the **Start Page** is. When you develop an application, you'll spend most of your time in this central area.



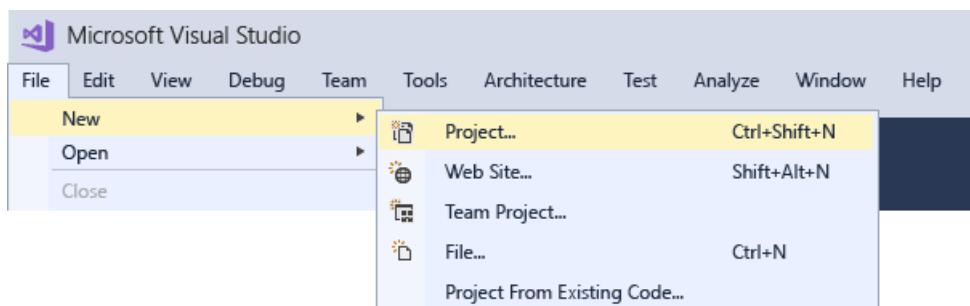
Create a simple application

Create the project

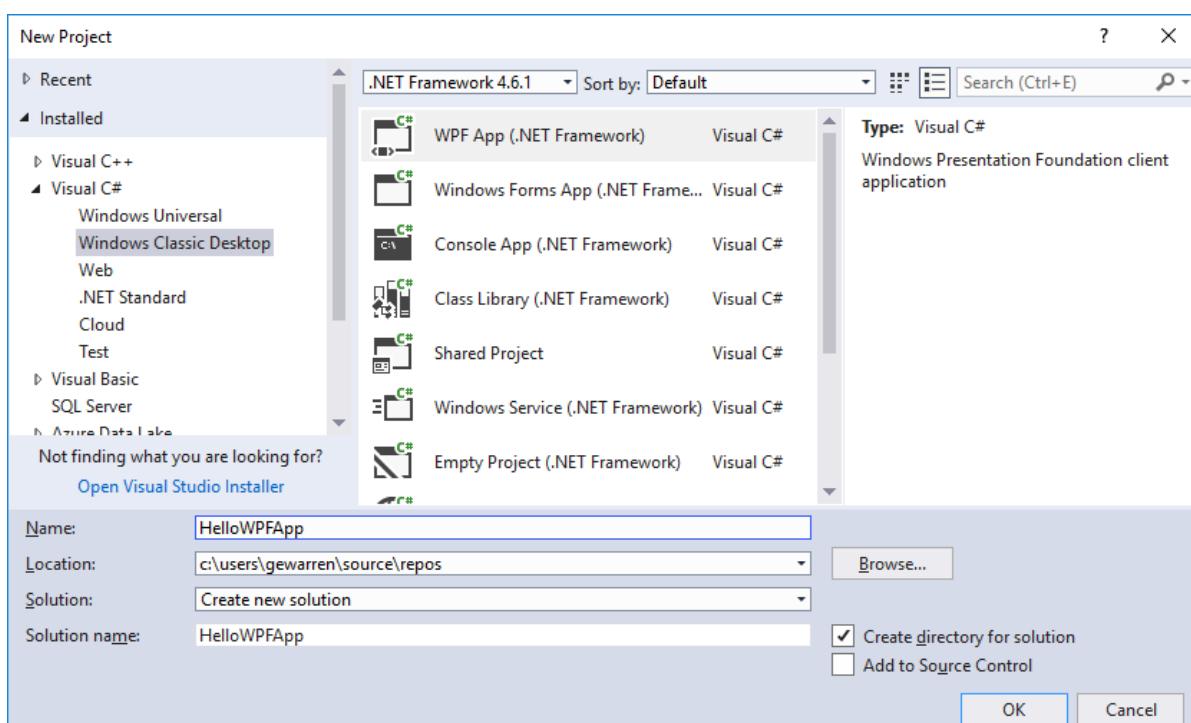
When you create an application in Visual Studio, you first create a project and a solution. For this example, you'll create a Windows Presentation Foundation (WPF) project.

To create the WPF project

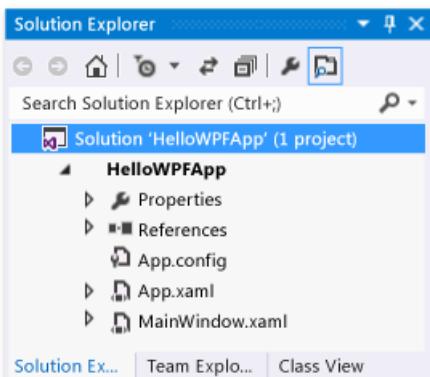
1. Create a new project. On the menu bar, choose **File, New, Project....**



2. Choose the Visual Basic or the Visual C# WPF App template by choosing in the left pane **Installed, Visual C#, Windows Classic Desktop**, for example, and then choosing **WPF App (.NET Framework)** in the middle pane. Name the project HelloWPFApp at the bottom of the New Project dialog.



Visual Studio creates the HelloWPFApp project and solution, and **Solution Explorer** shows the various files. The WPF Designer shows a design view and a XAML view of MainWindow.xaml in a split view. You can slide the splitter to show more or less of either view. You can choose to see only the visual view or only the XAML view. (For more information, see [WPF Designer for Windows Forms Developers](#).) The following items appear in **Solution Explorer**:

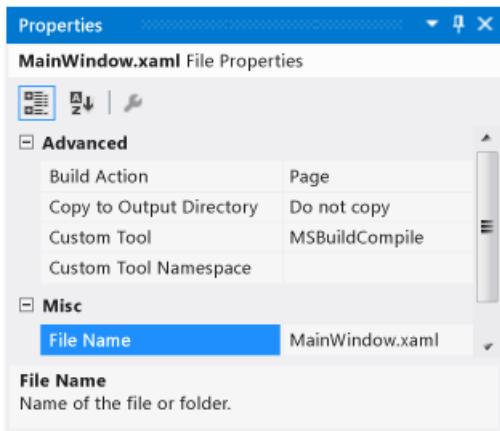


After you create the project, you can customize it. By using the **Properties** window (found on the **View** menu), you can display and change options for project items, controls, and other items in an application.

To change the name of MainWindow.xaml

Let's give MainWindow a more specific name.

1. In **Solution Explorer**, select MainWindow.xaml. You should see the **Properties** window, but if you don't, choose the **View** menu and then the **Properties Window** item.
2. Change the **File Name** property to `Greetings.xaml`.



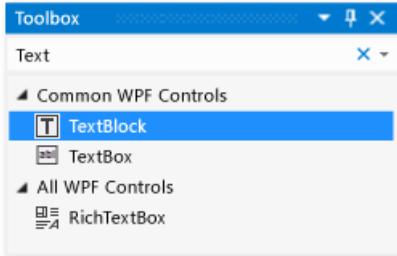
Solution Explorer shows that the name of the file is now Greetings.xaml, and the nested code file is now named Greetings.xaml.vb or Greetings.xaml.cs. This code file is nested under the .xaml file node to show they are closely related to each other.

Design the user interface (UI)

We will add three types of controls to this application: a TextBlock control, two RadioButton controls, and a Button control.

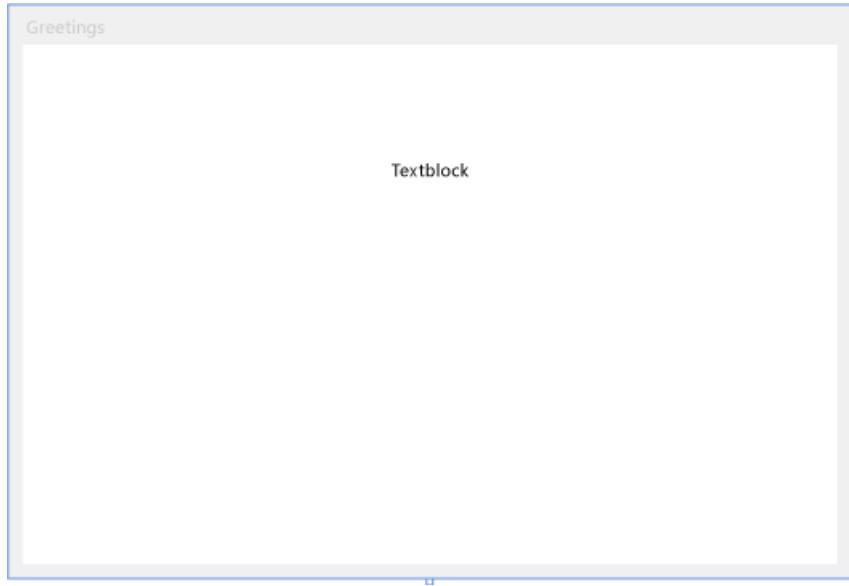
To add a TextBlock control

1. Open the **Toolbox** window by choosing the **View** menu and the **Toolbox** item.
2. In the **Toolbox**, expand the **Common WPF Controls** node to see the TextBlock control.



3. Add a TextBlock control to the design surface by choosing the **TextBlock** item and dragging it to the window on the design surface. Center the control near the top of the window.

Your window should resemble the following illustration:



The XAML markup should look something like the following:

```
<TextBlock HorizontalAlignment="Center" TextWrapping="Wrap" VerticalAlignment="Center" RenderTransformOrigin="4.08,2.312" Margin="237,57,221,238"><Run Text="TextBlock"/><InlineUIContainer><TextBlock TextWrapping="Wrap" Text="TextBlock"/>
```

To customize the text in the text block

1. In the XAML view, locate the markup for TextBlock and change the Text attribute:

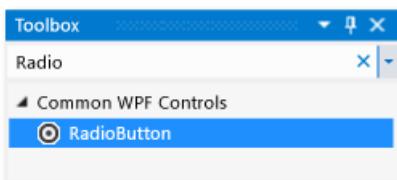
```
Text="Select a message option and then choose the Display button."
```

2. Re-center the TextBlock if necessary, and save your changes by pressing **Ctrl-s** or using the **File** menu item.

Next, you'll add two **RadioButton** controls to the form.

To add radio buttons

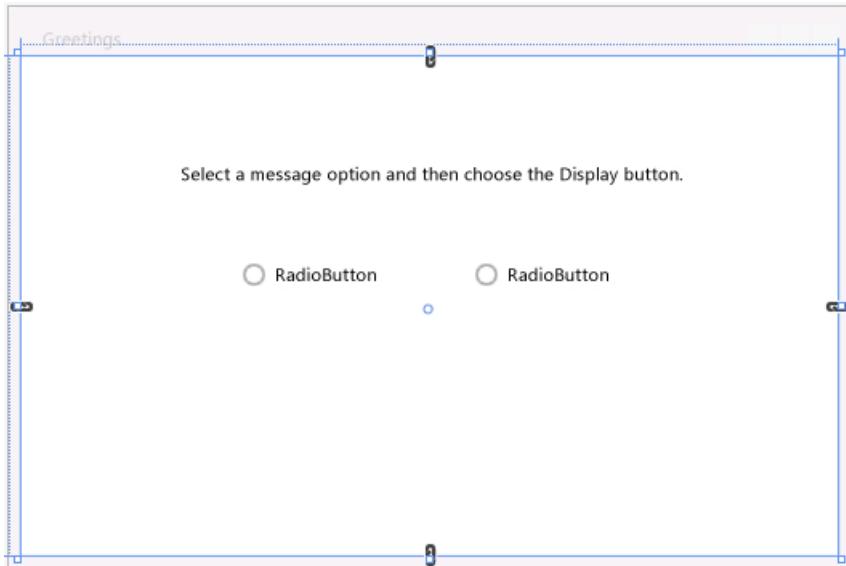
1. In the **Toolbox**, find the **RadioButton** control.



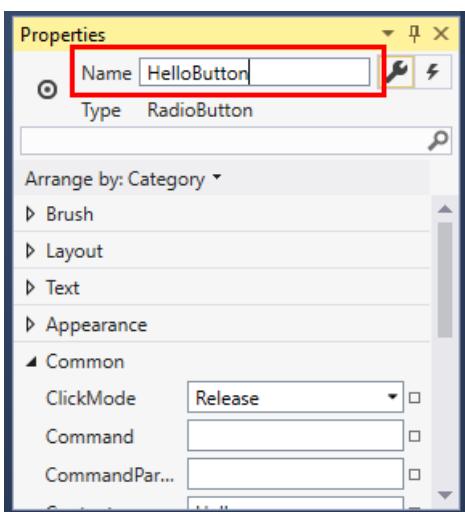
2. Add two RadioButton controls to the design surface by choosing the **RadioButton** item and dragging it to the window on the design surface. Move the buttons (by selecting them and using the arrow keys) so that

the buttons appear side by side under the **TextBlock** control.

Your window should look like this:



3. In the **Properties** window for the left RadioButton control, change the **Name** property (the property at the top of the **Properties** window) to **HelloButton**.



4. In the **Properties** window for the right RadioButton control, change the **Name** property to **GoodbyeButton**, and then save your changes.

You can now add display text for each RadioButton control. The following procedure updates the **Content** property for a RadioButton control.

To add display text for each radio button

1. On the design surface, open the shortcut menu for HelloButton by pressing the right mouse button on HelloButton, choose **Edit Text**, and then enter 'Hello'.
2. Open the shortcut menu for GoodbyeButton by pressing the right mouse button on GoodbyeButton, choose **Edit Text**, and then enter 'Goodbye'.

To set a radio button to be checked by default

In this step we'll set HelloButton to be checked by default so that one of the two radio buttons is always selected.

In the XAML view, locate the markup for HelloButton and add an **IsChecked** attribute:

```
IsChecked="True"
```

The final UI element that you'll add is a **Button** control.

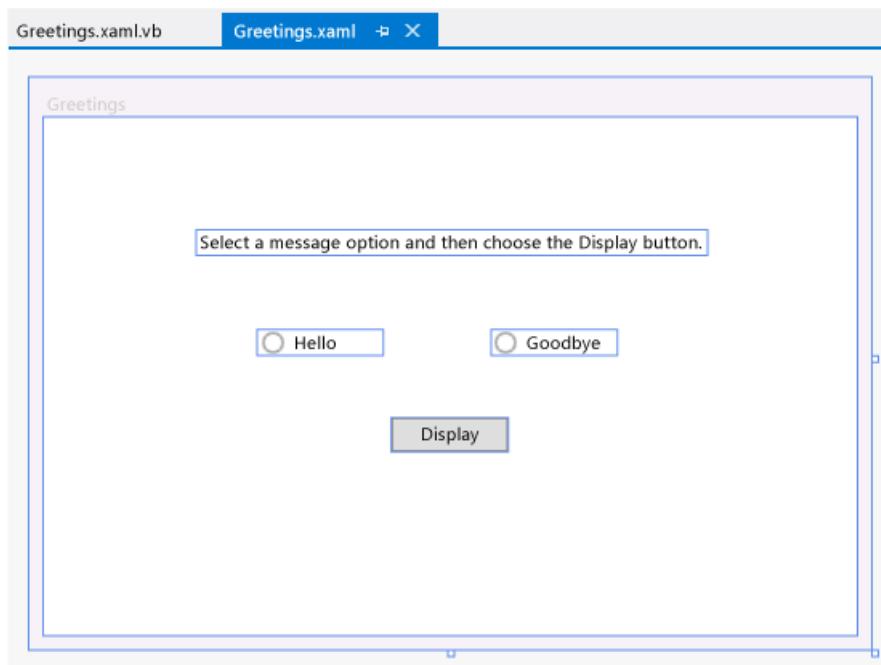
To add the button control

1. In the **Toolbox**, find the **Button** control, and then add it to the design surface under the **RadioButton** controls by dragging it to the form in the design view.
2. In the XAML view, change the value of **Content** for the Button control from `Content="Button"` to `Content="Display"`, and then save the changes.

The markup should resemble the following example:

```
<Button Content="Display" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75" Margin="215,204,0,0"/>
```

Your window should resemble the following illustration.



Add code to the Display Button

When this application runs, a message box appears after a user chooses a radio button and then chooses the **Display** button. One message box will appear for Hello, and another will appear for Goodbye. To create this behavior, you'll add code to the **Button_Click** event in `Greetings.xaml.vb` or `Greetings.xaml.cs`.

Add code to display message boxes

1. On the design surface, double-click the **Display** button.

`Greetings.xaml.vb` or `Greetings.xaml.cs` opens, with the cursor in the `Button_Click` event.

```
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)  
End Sub
```

```
private void Button_Click_1(object sender, RoutedEventArgs e)  
{  
}
```

2. Enter the following code:

```
If HelloButton.IsChecked = True Then
    MessageBox.Show("Hello.")
ElseIf GoodbyeButton.IsChecked = True Then
    MessageBox.Show("Goodbye.")
End If
```

```
if (HelloButton.IsChecked == true)
{
    MessageBox.Show("Hello.");
}
else if (GoodbyeButton.IsChecked == true)
{
    MessageBox.Show("Goodbye.");
}
```

3. Save the application.

Debug and test the application

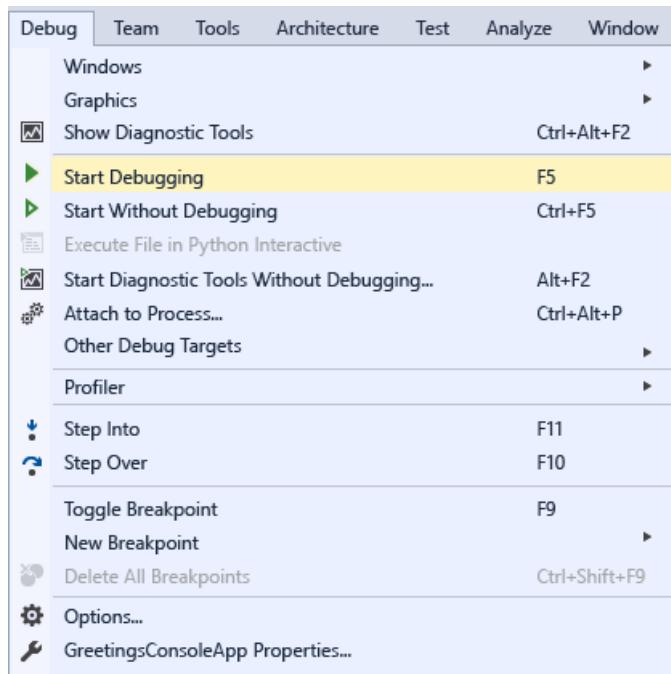
Next, you'll debug the application to look for errors and test that both message boxes appear correctly. The following instructions tell you how to build and launch the debugger, but later you might read [Building a WPF Application \(WPF\)](#) and [Debugging WPF](#) for more information.

Find and fix errors

In this step, you'll find the error that we caused earlier by changing the name of the MainWindow.xaml file.

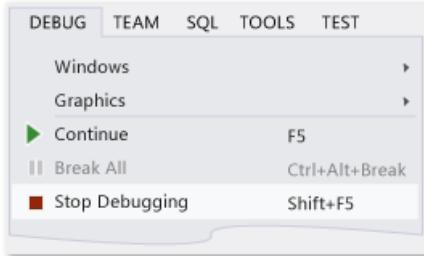
To start debugging and find the error

1. Start the debugger by selecting **Debug**, then **Start Debugging**.



A **Break Mode** window appears, and the **Output** window indicates that an IOException has occurred:
Cannot locate resource 'mainwindow.xaml'.

2. Stop the debugger by choosing **Debug, Stop Debugging**.



We renamed MainWindow.xaml to Greetings.xaml at the start of this walkthrough, but the code still refers to mainwindow.xaml as the startup URI for the application, so the project can't start.

To specify Greetings.xaml as the startup URI

1. In **Solution Explorer**, open the App.xaml file (in the C# project) or the Application.xaml file (in the Visual Basic project).
2. Change `StartupUri="MainWindow.xaml"` to `StartupUri="Greetings.xaml"`, and then save the changes.

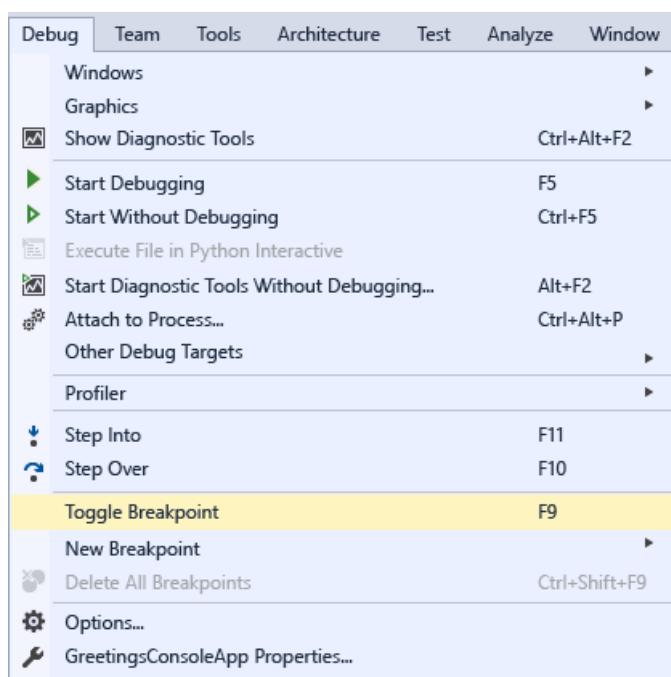
Start the debugger again (press **F5**). You should see the Greetings window of the application. Now close the application window to stop debugging.

To debug with breakpoints

You can test the code during debugging by adding some breakpoints. You can add breakpoints by choosing **Debug, Toggle Breakpoint**, by clicking in the left margin of the editor next to the line of code where you want the break to occur, or by pressing **F9**.

To add breakpoints

1. Open Greetings.xaml.vb or Greetings.xaml.cs, and select the following line: `MessageBox.Show("Hello.")`
2. Add a breakpoint from the menu by selecting **Debug**, then **Toggle Breakpoint**.



A red circle appears next to the line of code in the far left margin of the editor window.

3. Select the following line: `MessageBox.Show("Goodbye.")`.
4. Press the **F9** key to add a breakpoint, and then press **F5** to start debugging.
5. In the **Greetings** window, choose the **Hello** radio button, and then choose the **Display** button.

The line `MessageBox.Show("Hello.")` is highlighted in yellow. At the bottom of the IDE, the Autos, Locals, and

Watch windows are docked together on the left side, and the Call Stack, Breakpoints, Command, Immediate, and Output windows are docked together on the right side.

6. On the menu bar, choose **Debug, Step Out**.

The application resumes execution, and a message box with the word "Hello" appears.

7. Choose the **OK** button on the message box to close it.

8. In the **Greetings** window, choose the **Goodbye** radio button, and then choose the **Display** button.

The line `MessageBox.Show("Goodbye.")` is highlighted in yellow.

9. Choose the **F5** key to continue debugging. When the message box appears, choose the **OK** button on the message box to close it.

10. Close the application window to stop debugging.

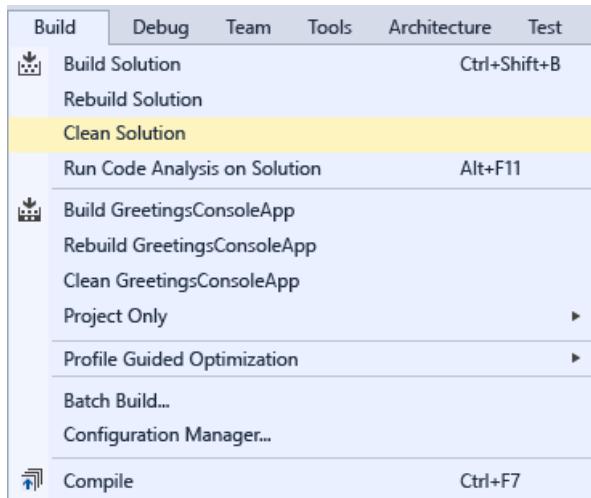
11. On the menu bar, choose **Debug, Disable All Breakpoints**.

Build a release version of the application

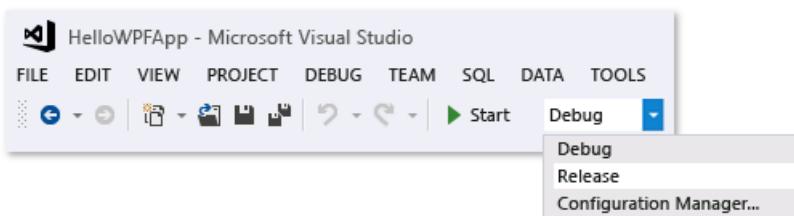
Now that you've verified that everything works, you can prepare a release build of the application.

To clean the solution files and build a release version

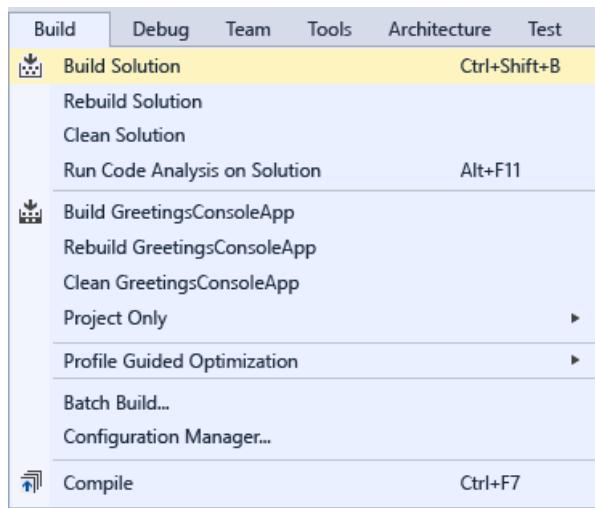
1. On the main menu, select **Build, Clean solution** to delete intermediate files and output files that were created during previous builds. This is not necessary, but it cleans up the debug build outputs.



2. Change the build configuration for HelloWPFApp from **Debug** to **Release** by using the dropdown control on the toolbar (it says "Debug" currently).



3. Build the solution by choosing **Build**, then **Build Solution**.



Congratulations on completing this walkthrough! You can find the .exe you built under your solution and project directory (...\\HelloWPFApp\\HelloWPFApp\\bin\\Release\\). If you want to explore more examples, see [Visual Studio Samples](#).

See also

- [What's New in Visual Studio 2017](#)
- [Get Started Developing with Visual Studio](#)
- [Productivity Tips](#)

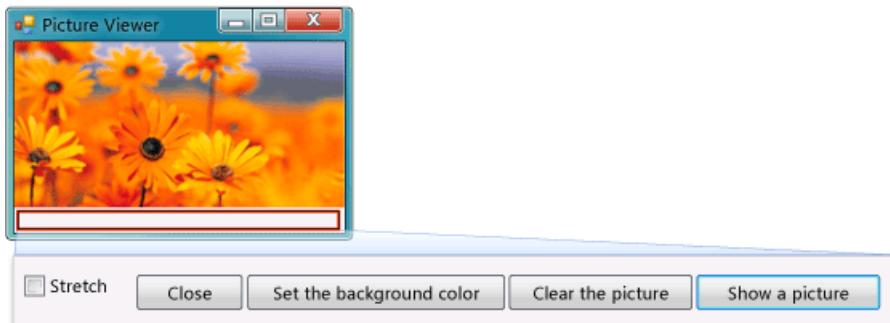
Tutorial 1: Create a Picture Viewer

12/22/2017 • 2 min to read • [Edit Online](#)

In this tutorial, you build a program that loads a picture from a file and displays it in a window. You learn how to drag controls like buttons and picture boxes on your form, set their properties, and use containers to smoothly resize the form. You also get started writing code. You learn how to:

- Create a new project.
- Test (debug) an application.
- Add basic controls like check boxes and buttons to a form.
- Position controls on a form using layouts.
- Add **Open File** and **Color** dialog boxes to a form.
- Write code using IntelliSense and code snippets.
- Write event handler methods.

When you finish, your program will look like the following picture.



Picture that you create in this tutorial

To download a completed version of the sample, see [Complete Picture Viewer tutorial sample](#).

▶ For a video version of this topic, see [How Do I: Create a Picture Viewer in Visual Basic?](#) or [How Do I: Create a Picture Viewer in C#?](#).

NOTE

These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio. Visual C# and Visual Basic are both covered in this tutorial, so focus on information specific to the programming language that you're using.

To see code for Visual Basic, choose the **VB** tab at the top of code blocks, and to see code for Visual C#, choose the **C#** tab. If you're interested in learning about Visual C++, see [Getting Started](#) and [C++ Language Tutorial](#).

If you're interested in learning how to write Visual C# or Visual Basic UWP apps, see [Build UWP apps](#).

Related Topics

| TITLE | DESCRIPTION |
|--|--|
| Step 1: Create a Windows Forms Application Project | Begin by creating a Windows Forms application project. |
| Step 2: Run Your Program | Run the Windows Forms application program that you created in the previous step. |
| Step 3: Set Your Form Properties | Change the way your form looks using the Properties window. |
| Step 4: Lay Out Your Form with a TableLayoutPanel Control | Add a <code>TableLayoutPanel1</code> control to your form. |
| Step 5: Add Controls to Your Form | Add controls, such as a <code>PictureBox</code> control and a <code>CheckBox</code> control, to your form. Add buttons to your form. |
| Step 6: Name Your Button Controls | Rename your buttons to something more meaningful. |
| Step 7: Add Dialog Components to Your Form | Add an <code>OpenFileDialog</code> component and a <code>ColorDialog</code> component to your form. |
| Step 8: Write Code for the Show a Picture Button Event Handler | Write code using the IntelliSense tool. |
| Step 9: Review, Comment, and Test Your Code | Review and test your code. Add comments as needed. |
| Step 10: Write Code for Additional Buttons and a Check Box | Write code to make other buttons and a check box work using IntelliSense. |
| Step 11: Run Your Program and Try Other Features | Run your program and set the background color. Try other features, such as changing colors, fonts, and borders. |

Step 1: Create a Windows Forms Application Project

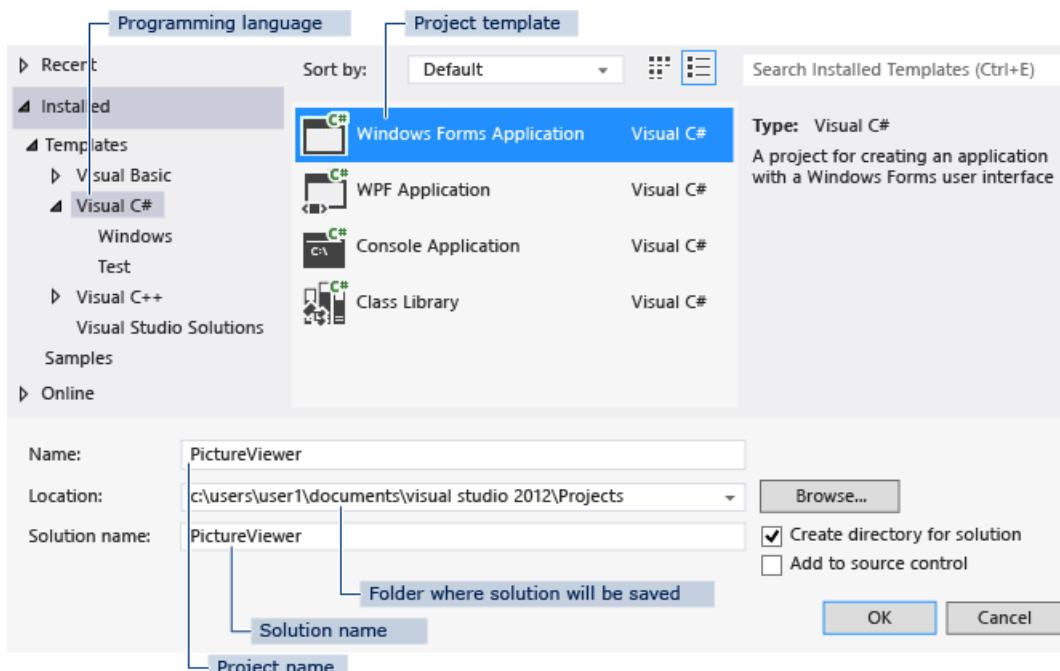
12/22/2017 • 3 min to read • [Edit Online](#)

When you create a picture viewer, the first step is to create a Windows Forms Application project.

For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 1](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 1](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

To create a Windows Forms Application project

1. On the menu bar, choose **File**, **New**, **Project**. The dialog box should look like this.



New project dialog box

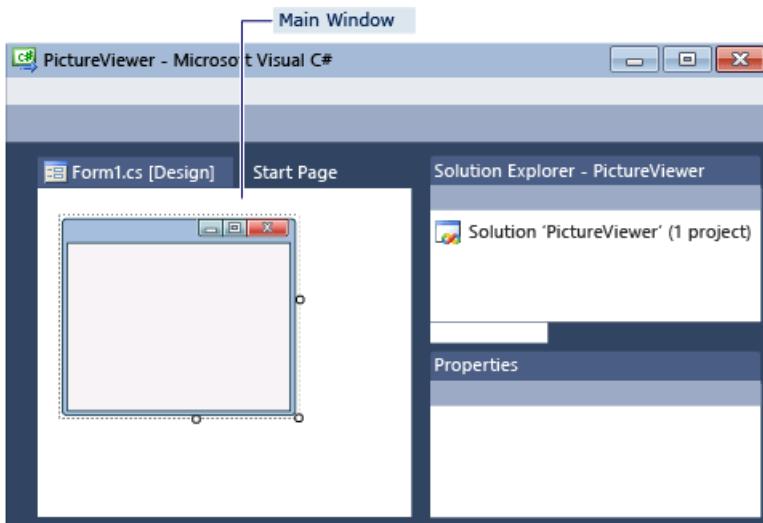
2. Choose either **Visual C#** or **Visual Basic** in the **Installed Templates** list.
3. In the templates list, choose the **Windows Forms Application** icon. Name the new form **PictureViewer**, and then choose the **OK** button.

Visual Studio creates a solution for your program. A solution acts as a container for all of the projects and files needed by your program. These terms will be explained in more detail later in this tutorial.

4. The following illustration shows what you should now see in the Visual Studio interface.

NOTE

Your window layout may not look exactly like this illustration. The precise window layout depends on the version of Visual Studio, the programming language you are using, and other factors. However, you should verify that all three windows appear.



IDE window

The interface contains three windows: a main window, **Solution Explorer**, and the **Properties** window.

If any of these windows are missing, restore the default window layout by, on the menu bar, choosing **Window, Reset Window Layout**. You can also display windows by using menu commands. On the menu bar, choose **View, Properties Window** or **Solution Explorer**. If any other windows are open, close them by choosing the **Close** (x) button in their upper-right corners.

5. The illustration shows the following windows (going clockwise from the upper-left corner):

- **Main window** In this window, you'll do most of your work, such as working with forms and editing code. In the illustration, the window shows a form in the Form Editor. At the top of the window, the **Start Page** tab and the **Form1.cs [Design]** tab appear. (In Visual Basic, the tab name ends with .vb instead of .cs.)
- **Solution Explorer window** In this window, you can view and navigate to all items in your solution. If you choose a file, the contents of the **Properties** window changes. If you open a code file (which ends in .cs in Visual C# and .vb in Visual Basic), the code file or a designer for the code file appears. A designer is a visual surface onto which you can add controls such as buttons and lists. For Visual Studio forms, the designer is called the Windows Forms Designer.
- **Properties window** In this window, you can change the properties of items that you choose in the other windows. For example, if you choose Form1, you can change its title by setting the **Text** property, and you can change the background color by setting the **Backcolor** property.

NOTE

The top line in **Solution Explorer** shows **Solution 'PictureViewer' (1 project)**, which means that Visual Studio created a solution for you. A solution can contain more than one project, but for now, you'll work with solutions that contain only one project.

6. On the menu bar, choose **File, Save All**.

As an alternative, choose the **Save All** button on the toolbar, which the following illustration shows.



Save All toolbar button

Visual Studio automatically fills in the folder name and the project name and then saves the project in your projects folder.

To continue or review

- To go to the next tutorial step, see [Step 2: Run Your Program](#).
- To return to the overview topic, see [Tutorial 1: Create a Picture Viewer](#).

Step 2: Run Your Program

12/22/2017 • 1 min to read • [Edit Online](#)

When you created a new solution, you actually built a program that runs. It doesn't do much yet—it just displays an empty window that shows **Form1** in the title bar. But it does run, as you're about to find out.

For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 1](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 1](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

To run your program

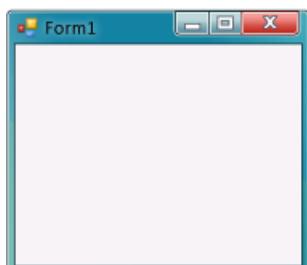
1. Use one of the following methods to run your program.

- Choose the **F5** key.
- On the menu bar, choose **Debug, Start Debugging**.
- On the toolbar, choose the **Start Debugging** button, which appears as follows.



Start Debugging toolbar button

2. Visual Studio runs your program, and a window called **Form1** appears. The following diagram shows the program you just built. The program is running, and you will soon add to it.



Windows Form Application program running

3. Go back to the Visual Studio integrated development environment (IDE), and look at the new toolbar.

Additional buttons appear on the toolbar when you run a program. These buttons let you do things like stop and start your program, and help you track down any errors (bugs) it may have. For this example, we're just using it to start and stop the program.



Debugging toolbar

4. Use one of the following methods to stop your program.

- On the toolbar, choose the **Stop Debugging** button.
- On the menu bar, choose **Debug, Stop Debugging**.
- Choose the X button in the upper corner of the **Form1** window.

NOTE

When you run your program from inside the IDE, it's called *debugging* because you typically do it to locate and fix bugs (errors) in the program. Although this program is small and doesn't really do anything yet, it's still a real program. You follow the same procedure to run and debug other programs. To learn more about debugging, see [Debugger Basics](#).

To continue or review

- To go to the next tutorial step, see [Step 3: Set Your Form Properties](#).
- To return to the previous tutorial step, see [Step 1: Create a Windows Forms Application Project](#).

Step 3: Set Your Form Properties

12/22/2017 • 2 min to read • [Edit Online](#)

Next, you use the **Properties** window to change the way your form looks.

For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 1](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 1](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

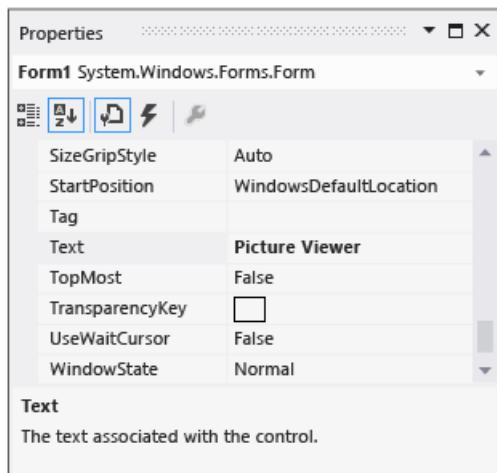
To set your form properties

1. Be sure you're looking at Windows Forms Designer. In the Visual Studio integrated development environment (IDE), choose the **Form1.cs [Design]** tab (or the **Form1.vb [Design]** tab in Visual Basic).
2. Choose anywhere inside the form **Form1** to select it. Look at the **Properties** window, which should now be showing the properties for the form. Forms have various properties. For example, you can set the foreground and background color, title text that appears at the top of the form, size of the form, and other properties.

NOTE

If the **Properties** window doesn't appear, stop your program by choosing the square **Stop Debugging** button on the toolbar, or just close the window. If the program is stopped and you still don't see the **Properties** window, on the menu bar, choose **View, Properties Window**.

3. After the form is selected, find the **Text** property in the **Properties** window. Depending on how the list is sorted, you might need to scroll down. Choose **Text**, type **Picture Viewer**, and then choose ENTER. Your form should now have the text **Picture Viewer** in its title bar, and the **Properties** window should look similar to the following picture.



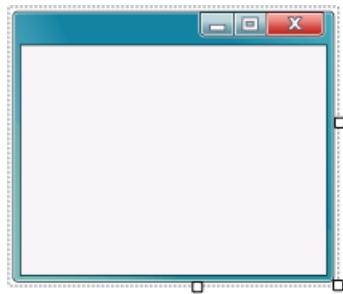
Properties window

NOTE

Properties can be ordered by a Categorized or Alphabetical view. You can switch between these two views by using the buttons on the **Properties** window. In this tutorial, it's easier to find properties through the Alphabetical view.

4. Go back to Windows Forms Designer. Choose the form's lower-right drag handle, which is the small white

square in the lower-right of the form and appears as follows.



Drag handle

Drag the handle to resize the form so the form is wider and a bit taller.

5. Look at the **Properties** window, and notice that the **Size** property has changed. The **Size** property changes each time you resize the form. Try dragging the form's handle to resize it to a form size of approximately 550, 350 (no need to be exact), which should work well for this project. As an alternative, you can enter the values directly in the **Size** property and then choose the ENTER key.
6. Run your program again. Remember, you can use any of the following methods to run your program.

- Choose the **F5** key.
- On the menu bar, choose **Debug, Start Debugging**.
- On the toolbar, choose the **Start Debugging** button, which appears as follows.



Start Debugging toolbar button

Just like before, the IDE builds and runs your program, and a window appears.

7. Before going to the next step, stop your program, because the IDE won't let you change your program while it's running. Remember, you can use any of the following methods to stop your program.

 - On the toolbar, choose the **Stop Debugging** button.
 - On the menu bar, choose **Debug, Stop Debugging**.
 - Choose the X button in the upper corner of the **Form1** window.

To continue or review

- To go to the next tutorial step, see [Step 4: Lay Out Your Form with a TableLayoutPanel Control](#).
- To return to the previous tutorial step, see [Step 2: Run Your Program](#).

Step 4: Lay Out Your Form with a TableLayoutPanel Control

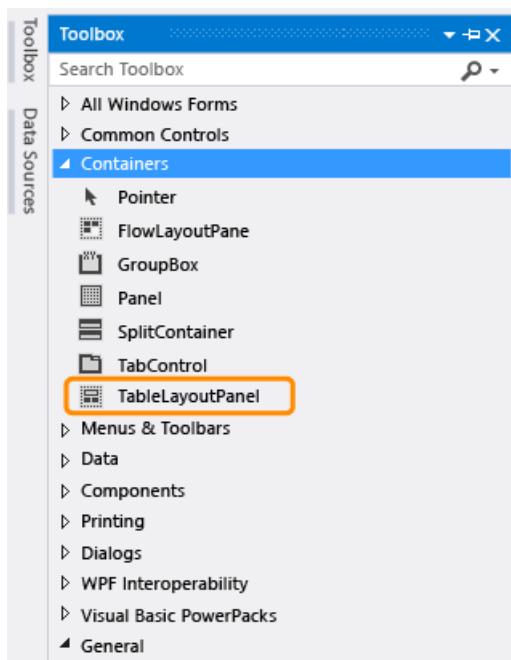
12/22/2017 • 4 min to read • [Edit Online](#)

In this step, you add a `TableLayoutPanel` control to your form. The TableLayoutPanel helps properly align controls in the form that you will add later.

▶ For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 2](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 2](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

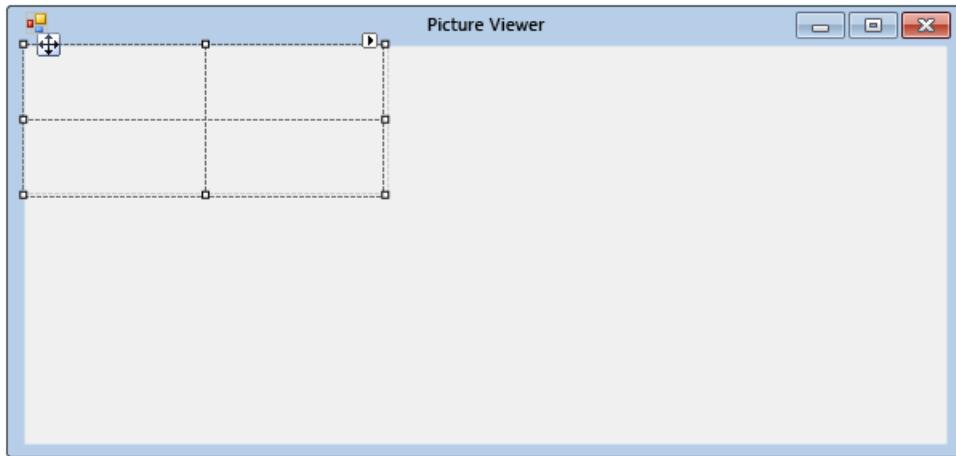
To lay out your form with a TableLayoutPanel control

1. On the left side of the Visual Studio IDE, locate the **Toolbox** tab. Choose the **Toolbox** tab, and the Toolbox appears. (Or, on the menu bar, choose **View, Toolbox**.)
2. Choose the small triangle symbol next to the **Containers** group to open it, as shown in the following picture.



Containers group

3. You can add controls like buttons, check boxes, and labels to your form. Double-click the `TableLayoutPanel1` control in the Toolbox. (Or, you can drag the control from the toolbox onto the form.) When you do this, the IDE adds a `TableLayoutPanel` control to your form, as shown in the following picture.



TableLayoutPanel control

NOTE

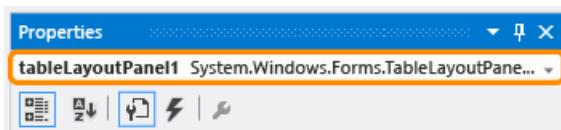
After you add your TableLayoutPanel, if a window appears inside your form with the title **TableLayoutPanel Tasks**, choose anywhere inside the form to close it. You will learn more about this window later in the tutorial.

Notice how the Toolbox expands to cover your form when you choose its tab, and closes after you choose anywhere outside of it. That's the IDE auto-hide feature. You can turn it on or off for any of the windows by choosing the pushpin icon in the upper-right corner of the window to toggle auto-hide and lock it in place. The pushpin icon appears as follows.



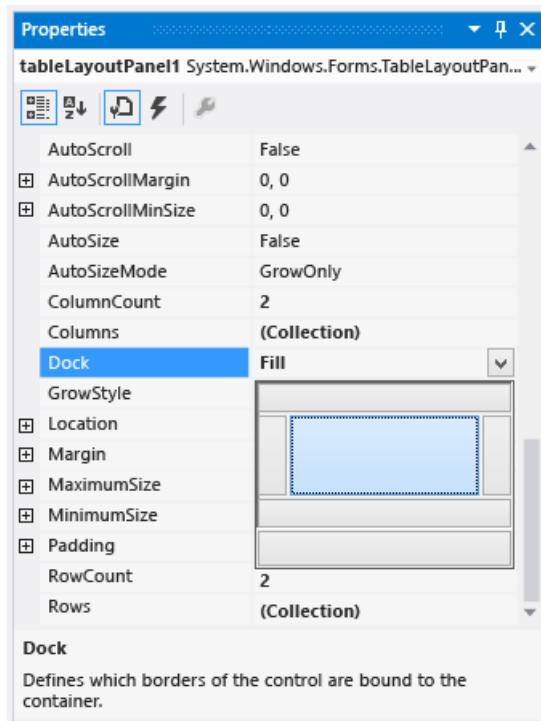
Pushpin icon

4. Be sure **TableLayoutPanel** is selected by choosing it. You can verify what control is selected by looking at the drop-down list at the top of the **Properties** window, as shown in the following picture.



Properties window showing TableLayoutPanel control

5. Choose the **Alphabetical** button on the toolbar in the **Properties** window. This causes the list of properties in the **Properties** window to display in alphabetical order, which will make it easier to locate properties in this tutorial.
6. The control selector is a drop-down list at the top of the **Properties** window. In this example, it shows that a control called `tableLayoutPanel1` is selected. You can select controls either by choosing an area in Windows Forms Designer or by choosing from the control selector. Now that `TableLayoutPanel1` is selected, find the **Dock** property and choose **Dock**, which should be set to **None**. Notice that a drop-down arrow appears next to the value. Choose the arrow, and then select the **Fill** button (the large button in the middle), as shown in the following picture.



Properties window with Fill selected

Docking in Visual Studio refers to when a window is attached to another window or area in the IDE. For example, the Properties window can be undocked - that is, unattached and free-floating within Visual Studio - or it can be docked against **Solution Explorer**.

- After you set the TableLayoutPanel **Dock** property to **Fill**, the panel fills the entire form. If you resize the form again, the TableLayoutPanel stays docked, and resizes itself to fit.

NOTE

A TableLayoutPanel works like a table in Microsoft Office Word: It has rows and columns, and an individual cell can span multiple rows and columns. Each cell can hold one control (like a button, a check box, or a label). Your TableLayoutPanel will have a **PictureBox** control spanning its entire top row, a **CheckBox** control in its lower-left cell, and four **Button** controls in its lower-right cell.

- Currently, the TableLayoutPanel has two equal-size rows and two equal-size columns. You need to resize them so the top row and right column are both much bigger. In Windows Forms Designer, select the TableLayoutPanel. In the upper-right corner, there is a small black triangle button, which appears as follows.



Triangle button

This button indicates that the control has tasks that help you set its properties automatically.

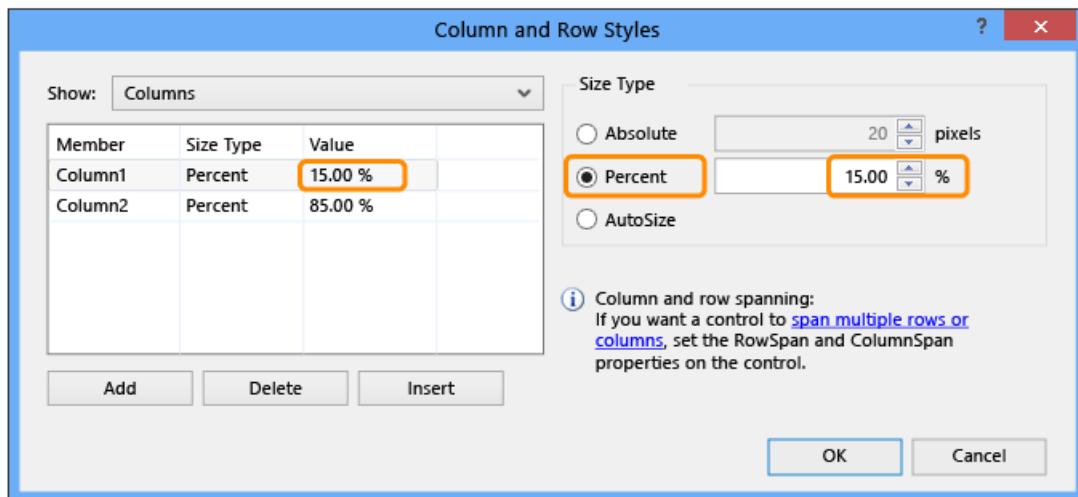
- Choose the triangle to display the control's task list, as shown in the following picture.



TableLayoutPanel tasks

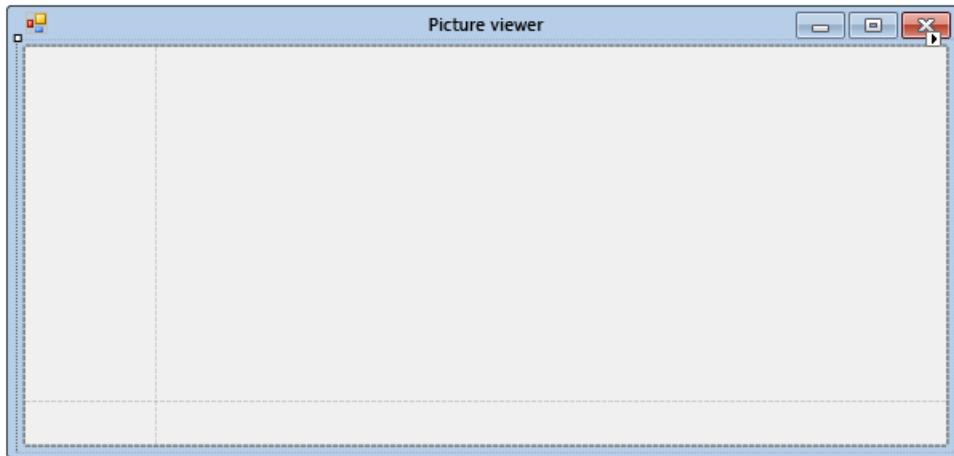
- Choose the **Edit Rows and Columns** task to display the **Column and Row Styles** window. Choose

Column1, and set its size to 15 percent by being sure the **Percent** button is selected and entering **15** in the **Percent** box. (That's a **NumericUpDown** control, which you will use in a later tutorial.) Choose **Column2** and set it to 85 percent. Don't choose the **OK** button yet, because the window will close. (But if you do, you can reopen it using the task list.)



TableLayoutPanel column and row styles

11. From the **Show** drop-down list at the top of the window, choose **Rows**. Set **Row1** to 90 percent and **Row2** to 10 percent.
12. Choose the **OK** button. Your TableLayoutPanel should now have a large top row, a small bottom row, a small left column, and a large right column. You can resize the rows and columns in the TableLayoutPanel by choosing **tableLayoutPanel1** in the form and then dragging its row and column borders.



Form1 with resized TableLayoutPanel

To continue or review

- To go to the next tutorial step, see [Step 5: Add Controls to Your Form](#).
- To return to the previous tutorial step, see [Step 3: Set Your Form Properties](#).

Step 5: Add Controls to Your Form

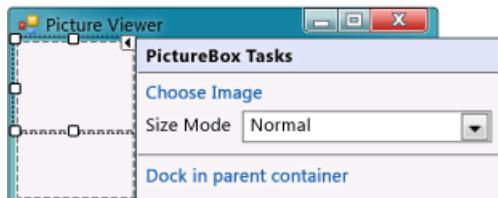
12/22/2017 • 5 min to read • [Edit Online](#)

In this step, you add controls, such as a **PictureBox** control and a **checkbox** control, to your form. You then add buttons to your form.

For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 2](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 2](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

To add controls to your form

1. Go to the Toolbox tab (located on the left side of the Visual Studio IDE) and expand the **Common Controls** group. This shows the most common controls that you see on forms.
2. Choose the TableLayoutPanel control on the form. To verify that the TableLayoutPanel is selected, make sure that its name appears in the dropdown list box at the top of the **Properties** window. You can also choose form controls by using the dropdown list box at the top of the **Properties** window. Choosing a control this way can often be easier than choosing a tiny control with a mouse.
3. Double-click the **PictureBox** item to add a PictureBox control to your form. Because the TableLayoutPanel is docked to fill your form, the IDE adds the PictureBox control to the first empty cell (the upper left corner).
4. Choose the new PictureBox control to select it, and then choose the black triangle on the new PictureBox control to display its task list, as shown in the following picture.



PictureBox tasks

NOTE

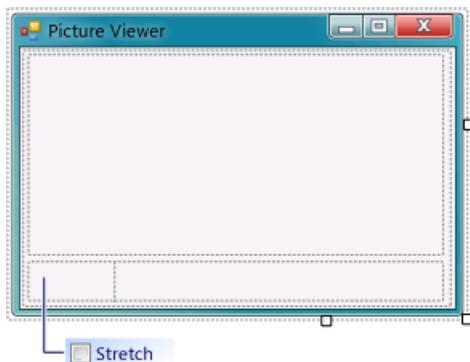
If you accidentally add the wrong type of control to your TableLayoutPanel, you can delete it. Right-click the control, and then choose **Delete** on its context menu. You can also remove controls from the form by using the menu bar. On the menu bar, choose **Edit, Undo**, or **Edit, Delete**.

5. Choose the **Dock in parent container** link. This automatically sets the PictureBox **Dock** property to **Fill**. To see this, choose the PictureBox control to select it, go to the **Properties** window, and be sure that the **Dock** property is set to **Fill**.
6. Make the PictureBox span both columns by changing its **ColumnSpan** property. Choose the PictureBox control and set its **ColumnSpan** property to **2**. Also, when the PictureBox is empty, you want to show an empty frame. Set its **BorderStyle** property to **Fixed3D**.

NOTE

If you don't see a **ColumnSpan** property for your PictureBox, then it's likely that the PictureBox was added to the form rather than the TableLayoutPanel. To fix this, choose the PictureBox, delete it, choose the TableLayoutPanel, and then add a new PictureBox.

7. Choose the TableLayoutPanel on the form and then add a **CheckBox** control to the form. Double-click the **CheckBox** item in the Toolbox to add a new CheckBox control to the next free cell in your table. Because a PictureBox takes up the first two cells in the TableLayoutPanel, the CheckBox control is added to the lower-left cell. Choose the **Text** property and type in the word **Stretch**, as shown in the following picture.



TextBox control with Stretch property

8. Choose the TableLayoutPanel on the form, and then go to the **Containers** group in the Toolbox (where you got your TableLayoutPanel control) and double-click the **FlowLayoutPanel** item to add a new control to the last cell in the PictureBox (bottom right). Then dock the FlowLayoutPanel in the TableLayoutPanel (either by choosing **Dock in parent container** on the FlowLayoutPanel's black triangle task list, or by setting the FlowLayoutPanel's **Dock** property to **Fill**).

NOTE

A FlowLayoutPanel is a container that arranges other controls in neat rows in order. When you resize a FlowLayoutPanel, if it has room to lay out all of its controls in a single row, it does that. Otherwise, it arranges them in lines, one on top of the other. You will use a FlowLayoutPanel to hold four buttons. If the buttons arrange one on top another when added, be sure that the FlowLayoutPanel is selected before adding the buttons. Although it was stated earlier that each cell can hold only one control, the lower-right cell of the TableLayoutPanel has four button controls. This is because you can put a control in a cell that holds other controls. That kind of control is called a container, and the FlowLayoutPanel is a container.

To add buttons

1. Choose the new FlowLayoutPanel that you added. Go to **Common Controls** in the Toolbox and double-click the **Button** item to add a button control called **button1** to your FlowLayoutPanel. Repeat to add another button. The IDE determines that there's already a button called **button1** and calls the next one **button2**.
2. Typically, you add the other buttons using the Toolbox. This time, choose **button2**, and then on the menu bar, choose **Edit, Copy** (or press Ctrl+C). On the menu bar, choose **Edit, Paste** (or press Ctrl+V) to paste a copy of your button. Now paste it again. The IDE has now added **button3** and **button4** to the FlowLayoutPanel.

NOTE

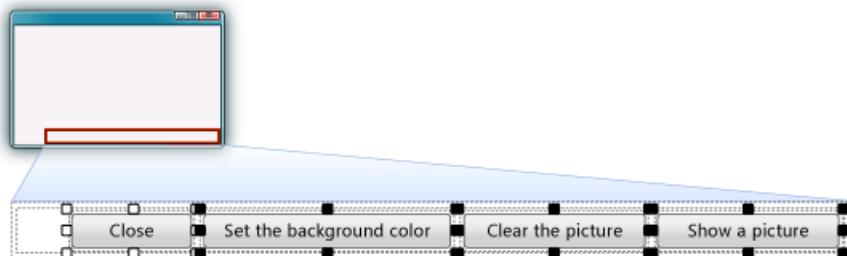
You can copy and paste any control. The IDE names and places the new controls in a logical manner. If you paste a control into a container, the IDE chooses the next logical space for placement.

3. Choose the first button and set its **Text** property to **Show a picture**. Then set the **Text** properties of the next three buttons to **Clear the picture**, **Set the background color**, and **Close**.
4. The next step is to size the buttons and arrange them so they align to the right side of the panel. Choose the FlowLayoutPanel and look at its **FlowDirection** property. Change it so it's set to **RightToLeft**. As soon as you do, the buttons should align themselves to the right side of the cell, and reverse their order so that the **Show a picture** button is on the right.

NOTE

If the buttons are still in the wrong order, you can drag the buttons around the FlowLayoutPanel to rearrange them in any order. You can choose a button and drag it left or right.

5. Choose the **Close** button to select it. Hold down the CTRL key and choose the other three buttons, so that they are all selected. While all the buttons are selected, go to the **Properties** window and scroll up to the **AutoSize** property. This property tells the button to automatically resize itself to fit all of its text. Set it to **true**. Your buttons should now be sized properly and be in the right order. (As long as all four buttons are selected, you can change all four **AutoSize** properties at the same time.) The following picture shows the four buttons.



Picture Viewer with four buttons

6. Now run your program again to see your newly laid out form. Choosing the buttons and the check box doesn't do anything yet, but it will work soon.

To continue or review

- To go to the next tutorial step, see [Step 6: Name Your Button Controls](#).
- To return to the previous tutorial step, see [Step 4: Lay Out Your Form with a TableLayoutPanel Control](#).

Step 6: Name Your Button Controls

2/6/2018 • 5 min to read • [Edit Online](#)

There's only one PictureBox on your form. When you added it, the IDE automatically named it **pictureBox1**. There's only one CheckBox, which is named **checkBox1**. Soon, you will write some code, and that code will refer to the CheckBox and PictureBox. Because there's only one of each of these controls, you will know what it means when you see **pictureBox1** or **checkBox1** in your code.

NOTE

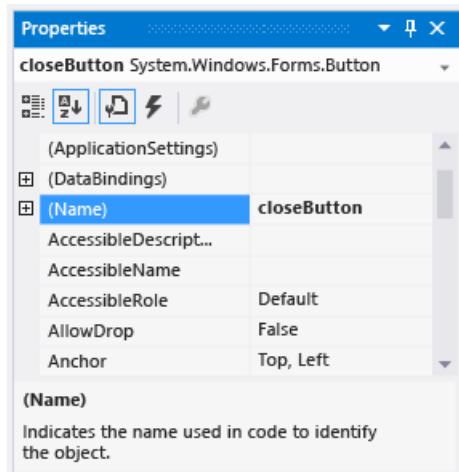
In Visual Basic, the default first letter of any control name is initial cap, so the names are **PictureBox1**, **CheckBox1**, and so on.

There are four buttons on your form, and the IDE named them **button1**, **button2**, **button3**, and **button4**. By just looking at their current names, you don't know which button is the **Close** button and which one is the **Show a picture** button. That's why giving your button controls more informative names is helpful.

For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 3](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 3](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

To name your button controls

1. On the form, choose the **Close** button. (If you still have all the buttons selected, choose the ESC key to cancel the selection.) Scroll in the **Properties** window until you see the **(Name)** property. (The **(Name)** property is near the top when the properties are alphabetical.) Change the name to **closeButton**, as shown in the following picture.



Properties window with closeButton name

NOTE

If you try changing the name of your button to **closeButton**, with a space between the words close and Button, the IDE displays an error message: "Property value is not valid." Spaces (and a few other characters) are not allowed in control names.

2. Rename the other three buttons to **backgroundButton**, **clearButton**, and **showButton**. You can verify the

names by choosing the control selector drop-down list in the **Properties** window. The new button names appear.

- Double-click the **Show a picture** button on the form. As an alternative, choose the **Show a picture** button on the form, and then choose the ENTER key. When you do, the IDE opens an additional tab in the main window called **Form1.cs (Form1.vb)** if you're using Visual Basic). This tab shows the code file behind the form, as shown in the following picture.

```
Form1.cs  Form1.cs [Design]*  
Picture_Viewer.Form1  showButton_Click(object sender, EventArgs e)  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
  
namespace Picture_Viewer  
{  
    public partial class Form1 : Form  
    {  
        public Form1()  
        {  
            InitializeComponent();  
        }  
  
        private void tableLayoutPanel1_Paint(object sender, PaintEventArgs e)  
        {  
        }  
  
        private void showButton_Click(object sender, EventArgs e)  
        {  
        }  
    }  
}
```

Form1.cs tab with Visual C# code

- Focus on this part of the code. (Choose the **VB** tab below if you're using Visual Basic to view the Visual Basic version of the code.)

```
Private Sub showButton_Click() Handles showButton.Click  
End Sub
```

```
private void showButton_Click(object sender, EventArgs e)  
{  
}
```

You are looking at code called `showButton_Click()`. The IDE added this to the form's code when you opened the code file for the **showButton** button. At design-time, when you open the code file for a control in a form, code is generated for the control if it doesn't already exist. This code, known as a *method*, runs when you run your program and choose the control - in this case, the **Show a picture** button.

NOTE

In this tutorial, the Visual Basic code that's automatically generated has been simplified by removing everything between the parentheses, `()`. Whenever this occurs, you can remove the same code. Your program will work either way. For the remainder of the tutorials, any automatically generated code is simplified whenever possible.

5. Choose the Windows Forms Designer tab again (**Form1.cs [Design]** in Visual C#, **Form1.vb [Design]** in Visual Basic) and then open the code file for the **Clear the picture** button to create a method for it in the form's code. Repeat this for the remaining two buttons. Each time, the IDE adds a new method to the form's code file.
6. To add one more method, open the code file for the CheckBox control in Windows Forms Designer to make the IDE add a `checkBox1_CheckedChanged()` method. That method is called whenever the user selects or clears the check box.

NOTE

When working on a program, you often move between the code editor and Windows Forms Designer. The IDE makes it easy to navigate in your project. Use **Solution Explorer** to open Windows Forms Designer by double-clicking **Form1.cs** in Visual C# or **Form1.vb** in Visual Basic, or on the menu bar, choose **View, Designer**.

The following shows the new code that you see in the code editor.

```
Private Sub clearButton_Click() Handles clearButton.Click
End Sub

Private Sub backgroundButton_Click() Handles backgroundButton.Click
End Sub

Private Sub closeButton_Click() Handles closeButton.Click
End Sub

Private Sub CheckBox1_CheckedChanged() Handles CheckBox1.CheckedChanged
End Sub
```

```
private void clearButton_Click(object sender, EventArgs e)
{
}

private void backgroundButton_Click(object sender, EventArgs e)
{
}

private void closeButton_Click(object sender, EventArgs e)
{
}

private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
}
```

The five methods that you added are called *event handlers*, because your program calls them whenever an event (like a user choosing a button or selecting a box) happens.

When you view the code for a control in the IDE at design time, Visual Studio adds an event handler method for the control if one isn't there. For example, when you double-click a button, the IDE adds an event handler for its Click event (which is called whenever the user chooses the button). When you double-click a check box, the IDE adds an event handler for its CheckedChanged event (which is called whenever the user selects or clears the box).

After you add an event handler for a control, you can return to it at any time from Windows Forms Designer

by double-clicking the control, or on the menu bar, choosing **View, Code**.

Names are important when you build programs, and methods (including event handlers) can have any name that you want. When you add an event handler with the IDE, it creates a name based on the control's name and the event being handled. For example, the Click event for a button named **showButton** is called the `showButton_Click()` event handler method. Also, opening and closing parentheses () are usually added after the method name to indicate that methods are being discussed. If you decide you want to change a code variable name, right-click the variable in the code and then choose **Refactor, Rename**. All instances of that variable in the code are renamed. See [Rename refactoring](#) for more information.

To continue or review

- To go to the next tutorial step, see [Step 7: Add Dialog Components to Your Form](#).
- To return to the previous tutorial step, see [Step 5: Add Controls to Your Form](#).

Step 7: Add Dialog Components to Your Form

12/22/2017 • 2 min to read • [Edit Online](#)

To enable your program to open picture files and choose a background color, in this step, you add an **OpenFileDialog** component and a **ColorDialog** component to your form.

A component is like a control in some ways. You use the Toolbox to add a component to your form, and you set its properties using the **Properties** window. But unlike a control, adding a component to your form doesn't add a visible item that the user can see on the form. Instead, it provides certain behaviors that you can trigger with code. It's a component that opens an **Open File** dialog box.

▶ For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 3](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 3](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

To add dialog components to your form

1. Choose the Windows Forms Designer (Form1.cs [Design] or Form1.vb [Design]) and then open the **Dialogs** group in the Toolbox.

NOTE

The **Dialogs** group in the Toolbox has components that open many useful dialog boxes for you, which can be used for opening and saving files, browsing folders, and choosing fonts and colors. You use two dialog components in this project: **OpenFileDialog** and **ColorDialog**.

2. To add a component called **openFileDialog1** to your form, double-click **OpenFileDialog**. To add a component called **colorDialog1** to your form, double-click **ColorDialog** in the Toolbox. (You use that one in the next tutorial step.) You should see an area at the bottom of Windows Forms Designer (beneath the Picture Viewer form) that has an icon for each of the two dialog components that you added, as shown in the following picture.



Dialog components

3. Choose the **openFileDialog1** icon in the area at the bottom of the Windows Forms Designer. Set two properties:

- Set the **Filter** property to the following (you can copy and paste it):

```
JPEG Files (*.jpg)|*.jpg|PNG Files (*.png)|*.png|BMP Files (*.bmp)|*.bmp>All files (*.*)|*.*
```

- Set the **Title** property to the following: **Select a picture file**

The **Filter** property settings specify the kinds of file types that will display in the **Select a picture** file dialog box.

NOTE

To see an example of the **Open File** dialog box in a different application, open Notepad or Paint, and on the menu bar, choose **File, Open**. Notice how there's a **Files of type** drop-down list at the bottom. You just used the **Filter** property in the **OpenFileDialog** component to set that up. Also, notice how the **Title** and **Filter** properties are bold in the **Properties** window. The IDE does that to show you any properties that have been changed from their default values.

To continue or review

- To go to the next tutorial step, see [Step 8: Write Code for the Show a Picture Button Event Handler](#).
- To return to the previous tutorial step, see [Step 6: Name Your Button Controls](#).

Step 8: Write Code for the Show a Picture Button Event Handler

12/22/2017 • 5 min to read • [Edit Online](#)

In this step, you make the **Show a picture** button work like this:

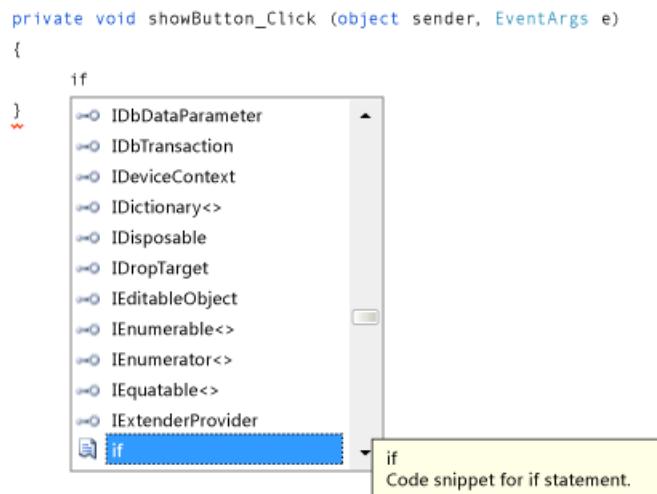
- When a user chooses that button, the program opens an **Open File** dialog box.
- If a user opens a picture file, the program shows that picture in the PictureBox.

The IDE has a powerful tool called IntelliSense that helps you write code. As you enter code, the IDE opens a box with suggested completions for partial words that you enter. It tries to determine what you want to do next, and automatically jumps to the last item you choose from the list. You can use the up or down arrows to move in the list, or you can keep typing letters to narrow the choices. When you see the choice you want, choose the TAB key to select it. Or, you can ignore the suggestions, if not needed.

For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 4](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 4](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

To write code for the Show a picture button event handler

1. Go to Windows Forms Designer and double-click the **Show a picture** button. The IDE immediately goes to the code designer and moves your cursor so it's inside the `showButton_Click()` method that you added previously.
2. Type an `i` on the empty line between the two braces `{ }`. (In Visual Basic, type on the empty line between Private Sub... and End Sub.) An **IntelliSense** window opens, as shown in the following picture.



IntelliSense with Visual C# code

3. The **IntelliSense** window should be highlighting the word **if**. (If not, enter a lowercase `f`, and it will.) Notice how a little *tooltip* box next to the **IntelliSense** window appears with the description, **Code snippet for if statement**. (In Visual Basic, the tooltip also states that this is a snippet, but with slightly different wording.) You want to use that snippet, so choose the TAB key to insert **if** into your code. Then choose the TAB key again to use the **if** snippet. (If you chose somewhere else and your **IntelliSense** window disappeared, backspace over the **i** and retype it, and the **IntelliSense** window opens again.)

```
private void showButton_Click(object sender, EventArgs e)
{
    if (true)
    {
    }
}
```

Visual C# code

4. Next, you use IntelliSense to enter more code to open an **Open File** dialog box. If the user chose the **OK** button, the PictureBox loads the file that the user selected. The following steps show how to enter the code, and although it's numerous steps, it's just a few keystrokes:

a. Start with the selected text **true** in the snippet. Type `op` to overwrite it. (In Visual Basic, you start with an initial cap, so type `Op`.)

b. The **IntelliSense** window opens and displays **openFileDialog1**. Choose the TAB key to select it. (In Visual Basic, it starts with an initial cap, so you see **OpenFileDialog1**. Ensure that **OpenFileDialog1** is selected.)

To learn more about `OpenFileDialog`, see [OpenFileDialog](#).

c. Type a period (`.`) (Many programmers call this a dot.) Because you typed a dot right after **openFileDialog1**, an **IntelliSense** window opens, filled in with all of the **OpenFileDialog** component's properties and methods. These are the same properties that appear in the **Properties** window when you choose it in Windows Forms Designer. You can also choose methods that tell the component to do things (like open a dialog box).

NOTE

The **IntelliSense** window can show you both properties and methods. To determine what is being shown, look at the icon on the left side of each item in the **IntelliSense** window. You see a picture of a block next to each method, and a picture of a wrench (or spanner) next to each property. There's also a lightning bolt icon next to each event. These pictures display as follows.



Method icon



Property icon



Event icon

- d. Start to type `ShowDialog` (capitalization is unimportant to IntelliSense). The `ShowDialog()` method will show the **Open File** dialog box. After the window has highlighted **ShowDialog**, choose the TAB key. You can also highlight "ShowDialog" and choose the F1 key to get help for it.

To learn more about the `ShowDialog()` method, see [ShowDialog Method](#).

- e. When you use a method on a control or a component (referred to as *calling a method*), you need to add parentheses. So enter opening and closing parentheses immediately after the "g" in `ShowDialog`: `()` It should now look like "openFileDialog1.ShowDialog()".

NOTE

Methods are an important part of any program, and this tutorial has shown several ways to use methods. You can call a component's method to tell it to do something, like how you called the **OpenFileDialog** component's `ShowDialog()` method. You can create your own methods to make your program do things, like the one you're building now, called the `showButton_Click()` method, which opens a dialog box and a picture when a user chooses a button.

- f. For Visual C#, add a space, and then add two equal signs (`==`). For Visual Basic, add a space, and then use a single equal sign (`=`). (Visual C# and Visual Basic use different equality operators.)
- g. Add another space. As soon as you do, another **IntelliSense** window opens. Start to type `DialogResult` and choose the TAB key to add it.

NOTE

When you write code to call a method, sometimes it returns a value. In this case, the **OpenFileDialog** component's `ShowDialog()` method returns a `DialogResult` value. `DialogResult` is a special value that tells you what happened in a dialog box. An **OpenFileDialog** component can result in the user choosing **OK** or **Cancel**, so its `ShowDialog()` method returns either `DialogResult.OK` or `DialogResult.Cancel`.

- h. Type a dot to open the `DialogResult` value **IntelliSense** window. Enter the letter `o` and choose the TAB key to insert **OK**.

To learn more about `DialogResult`, see [DialogResult](#).

NOTE

The first line of code should be complete. For Visual C#, it should be the following.

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
```

For Visual Basic, it should be the following.

```
If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
```

- i. Now add one more line of code. You can type it (or copy and paste it), but consider using IntelliSense to add it. The more familiar you are with IntelliSense, the more quickly you can write your own code. Your final `showButton_Click()` method looks like the following. (Choose the **VB** tab to view the Visual Basic version of the code.)

```
private void showButton_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        pictureBox1.Load(openFileDialog1.FileName);
    }
}
```

```
Private Sub showButton_Click() Handles showButton.Click
    If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
        PictureBox1.Load(OpenFileDialog1.FileName)
    End If

End Sub
```

To continue or review

- To go to the next tutorial step, see [Step 9: Review, Comment, and Test Your Code](#).
- To return to the previous tutorial step, see [Step 7: Add Dialog Components to Your Form](#).

Step 9: Review, Comment, and Test Your Code

12/22/2017 • 5 min to read • [Edit Online](#)

You next add a comment to your code. A comment is a note that doesn't change the way the program behaves. It makes it easier for someone who is reading your code to understand what it does. Adding comments to your code is a good habit to get into. In Visual C#, two forward slashes (//) mark a line as a comment. In Visual Basic, a single quotation mark ('') is used to mark a line as a comment. After you add a comment, you test your program. It's good practice to run and test your code frequently while you're working on your projects, so you can catch and fix any problems early, before the code gets more complicated. This is called *iterative testing*.

You just built something that works, and although it's not done yet, it can already load a picture. Before you add a comment to your code and test it, take time to review the code concepts, because you will use these concepts frequently:

- When you double-clicked the **Show a picture** button in Windows Forms Designer, the IDE automatically added a *method* to your program's code.
- Methods are how you organize your code: It's how your code is grouped together.
- Most of the time, a method does a small number of things in a specific order, like how your `showButton_Click()` method shows a dialog box and then loads a picture.
- A method is made up of code *statements*, or lines of code. Think of a method as a way to bundle code statements together.
- When a method is executed, or *called*, the statements in the method are executed in order, one after another, starting with the first one.

The following is an example of a statement.

```
pictureBox1.Load(openFileDialog1.FileName);
```

```
pictureBox1.Load(openFileDialog1.FileName)
```

Statements are what make your programs do things. In Visual C#, a statement always ends in a semicolon. In Visual Basic, the end of a line is the end of a statement. (No semicolon is needed in Visual Basic.) The preceding statement tells your `PictureBox` control to load the file that the user selected with the `OpenFileDialog` component.

 For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 5](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 5](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

To add comments

1. Add the following comment to your code.

```
Private Sub showButton_Click() Handles showButton.Click  
  
    ' Show the Open File dialog. If the user clicks OK, load the  
    ' picture that the user chose.  
    If OpenFileDialog1.ShowDialog() = DialogResult.OK Then  
        PictureBox1.Load(OpenFileDialog1.FileName)  
    End If  
  
End Sub
```

```
private void showButton_Click(object sender, EventArgs e)  
{  
    // Show the Open File dialog. If the user clicks OK, load the  
    // picture that the user chose.  
    if (openFileDialog1.ShowDialog() == DialogResult.OK)  
    {  
        pictureBox1.Load(openFileDialog1.FileName);  
    }  
}
```

NOTE

Your **showButton** button's Click event handler is now finished, and it works. You have started writing code, starting with an `if` statement. An `if` statement is how you tell your program, "Check this one thing, and if it's true, do these actions." In this case, you tell your program to open the **Open File** dialog box, and if the user selects a file and chooses the **OK** button, load that file in the PictureBox.

TIP

The IDE is built to make it easy for you to write code, and *code snippets* are one way it does that. A snippet is a shortcut that gets expanded into a small block of code.

You can see all of the snippets available. On the menu bar, choose **Tools, Code Snippets Manager**. For Visual C#, the `if` snippet is in **Visual C#**. For Visual Basic, the `if` snippets are in **Conditionals and Loops, Code Patterns**. You can use this manager to browse existing snippets or add your own snippets.

To activate a snippet when typing code, type it and choose the TAB key. Many snippets appear in the **IntelliSense** window, which is why you choose the TAB key twice: first to select the snippet from the **IntelliSense** window, and then to tell the IDE to use the snippet. (IntelliSense supports the `if` snippet, but not the `elseif` snippet.)

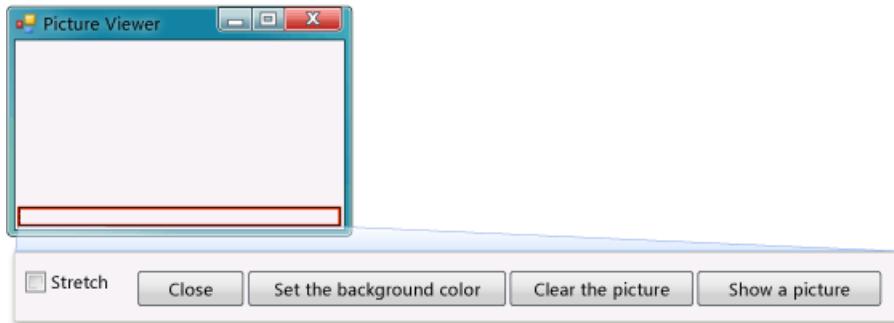
2. Before you run your program, save your program by choosing the **Save All** toolbar button, which appears as follows.



Save All button

Alternatively, to save your program, on the menu bar, choose **File, Save All**. It's a best practice to save early and often.

When it's running, your program should look like the following picture.



Picture Viewer

To test your program

1. Choose the F5 key or choose the **Start Debugging** toolbar button.
2. Choose the **Show a picture** button to run the code you just wrote. First, the program opens an **Open File** dialog box. Verify that your filters appear in the **Files of type** drop-down list at the bottom of the dialog box. Then navigate to a picture and open it. You can usually find sample pictures that ship with the Windows operating system in your **My Documents** folder, inside the **My Pictures\Sample Pictures** folder.

NOTE

If you don't see any images in the **Select a picture file** dialog box, be sure that the "All files (*.*)" filter is selected in the drop-down list on the lower right side of the dialog box.

3. Load a picture, and it appears in your PictureBox. Then try resizing your form by dragging its borders. Because you have your PictureBox docked inside a TableLayoutPanel, which itself is docked inside the form, your picture area will resize itself so that it's as wide as the form, and fills the top 90 percent of the form. That's why you used the TableLayoutPanel and FlowLayoutPanel containers: They keep your form sized correctly when the user resizes it.

Right now, larger pictures go beyond the borders of your picture viewer. In the next step, you'll add code to make pictures fit in the window.

To continue or review

- To go to the next tutorial step, see [Step 10: Write Code for Additional Buttons and a Check Box](#).
- To return to the previous tutorial step, see [Step 8: Write Code for the Show a Picture Button Event Handler](#).

Step 10: Write Code for Additional Buttons and a Check Box

12/22/2017 • 2 min to read • [Edit Online](#)

Now you're ready to complete the other four methods. You could copy and paste this code, but if you want to learn the most from this tutorial, type the code and use IntelliSense.

This code adds functionality to the buttons you added earlier. Without this code, the buttons don't do anything. The buttons use code in their `click` events (and the check box uses the `CheckChanged` event) to do different things when you activate the controls. For example, the `clearButton_Click` event, which activates when you choose the **Clear the picture** button, erases the current image by setting its `Image` property to `null` (or, `nothing`). Each event in the code includes comments that explain what the code does.

▶ For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 5](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 5](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

NOTE

As a best practice: Always comment your code. Comments are information for a person to read, and it's worth the time to make your code understandable. Everything on a comment line is ignored by the program. In Visual C#, you comment a line by typing two forward slashes at the beginning (//), and in Visual Basic you comment a line by starting with a single quotation mark (').

To write code for additional buttons and a check box

- Add the following code to your Form1 code file (Form1.cs or Form1.vb). Choose the **VB** tab to view Visual Basic code.

```

Private Sub clearButton_Click() Handles clearButton.Click
    ' Clear the picture.
    PictureBox1.Image = Nothing
End Sub

Private Sub backgroundButton_Click() Handles backgroundButton.Click
    ' Show the color dialog box. If the user clicks OK, change the
    ' PictureBox control's background to the color the user chose.
    If ColorDialog1.ShowDialog() = DialogResult.OK Then
        PictureBox1.BackColor = ColorDialog1.Color
    End If
End Sub

Private Sub closeButton_Click() Handles closeButton.Click
    ' Close the form.
    Close()
End Sub

Private Sub CheckBox1_CheckedChanged() Handles CheckBox1.CheckedChanged
    ' If the user selects the Stretch check box, change
    ' the PictureBox'sSizeMode property to "Stretch". If the user
    ' clears the check box, change it to "Normal".
    If CheckBox1.Checked Then
        PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
    Else
        PictureBox1.SizeMode = PictureBoxSizeMode.Normal
    End If
End Sub

```

```

private void clearButton_Click(object sender, EventArgs e)
{
    // Clear the picture.
    pictureBox1.Image = null;
}

private void backgroundButton_Click(object sender, EventArgs e)
{
    // Show the color dialog box. If the user clicks OK, change the
    // PictureBox control's background to the color the user chose.
    if (colorDialog1.ShowDialog() == DialogResult.OK)
        pictureBox1.BackColor = colorDialog1.Color;
}

private void closeButton_Click(object sender, EventArgs e)
{
    // Close the form.
    this.Close();
}

private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    // If the user selects the Stretch check box,
    // change the PictureBox's
    //SizeMode property to "Stretch". If the user clears
    // the check box, change it to "Normal".
    if (checkBox1.Checked)
        pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
    else
        pictureBox1.SizeMode = PictureBoxSizeMode.Normal;
}

```

To continue or review

- To go to the next tutorial step, see Step 11: Run Your Program and Try Other Features.

- To return to the previous tutorial step, see [Step 9: Review, Comment, and Test Your Code](#).

Step 11: Run Your Program and Try Other Features

12/22/2017 • 2 min to read • [Edit Online](#)

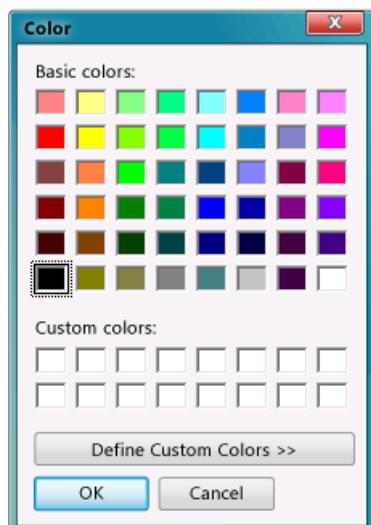
Your program is finished and ready to run. You can run your program and set the background color of the PictureBox. To learn more, try to improve the program by changing the color of the form, customizing the buttons and check box, and changing the properties of the form.

To download a completed version of the sample, see [Complete Picture Viewer tutorial sample](#).

▶ For a video version of this topic, see [Tutorial 1: Create a Picture Viewer in Visual Basic - Video 5](#) or [Tutorial 1: Create a Picture Viewer in C# - Video 5](#). These videos use an earlier version of Visual Studio, so there are slight differences in some menu commands and other user interface elements. However, the concepts and procedures work similarly in the current version of Visual Studio.

To run your program and set the background color

1. Choose F5, or on the menu bar, choose **Debug, Start Debugging**.
2. Before you open a picture, choose the **Set the background color** button. The **Color** dialog box opens.



Color dialog box

3. Choose a color to set the PictureBox background color. Look closely at the `backgroundButton_Click()` method to understand how it works.

NOTE

You can load a picture from the Internet by pasting its URL into the **Open File** dialog box. Try to find an image with a transparent background, so your background color shows.

4. Choose the **Clear the picture** button to make sure it clears. Then, exit the program by choosing the **Close** button.

To try other features

- Change the color of the form and the buttons by using the **BackColor** property.
- Customize your buttons and check box using the **Font** and **ForeColor** properties.
- Change your form's **FormBorderStyle** and **ControlBox** properties.

- Use your form's **AcceptButton** and **CancelButton** properties so that buttons are automatically chosen when the user chooses the ENTER or ESC key. Make the program open the **Open File** dialog box when the user chooses ENTER and close the box when the user chooses ESC.

To continue or review

- To learn more about programming in Visual Studio, see [Programming Concepts](#).
- To learn more about Visual Basic, see [Developing Applications with Visual Basic](#).
- To learn more about Visual C#, see [Introduction to the C# Language and the .NET Framework](#).
- To go to the next tutorial, see [Tutorial 2: Create a Timed Math Quiz](#).
- To return to the previous tutorial step, see [Step 10: Write Code for Additional Buttons and a Check Box](#).

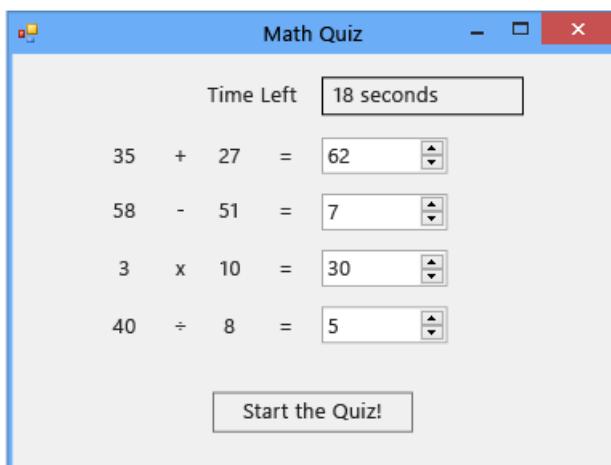
Tutorial 2: Create a Timed Math Quiz

12/22/2017 • 1 min to read • [Edit Online](#)

In this tutorial, you build a quiz in which the quiz taker must answer four random arithmetic problems within a specified time. You learn how to:

- Generate random numbers by using the `Random` class.
- Trigger events to occur at a specific time by using a **Timer** control.
- Control program flow by using `if else` statements.
- Perform basic arithmetic operations in code.

When you finish, your quiz will look like the following picture, except with different numbers.



Quiz that you create in this tutorial

To download a completed version of the quiz, see [Complete Math Quiz tutorial sample](#).

NOTE

This tutorial covers both Visual C# and Visual Basic, so focus on the information that's specific to the programming language that you're using.

Related Topics

| TITLE | DESCRIPTION |
|--|---|
| Step 1: Create a Project and Add Labels to Your Form | Start by creating the project, changing properties, and adding <code>Label</code> controls. |
| Step 2: Create a Random Addition Problem | Create an addition problem, and use the <code>Random</code> class to generate random numbers. |
| Step 3: Add a Countdown Timer | Add a countdown timer so that the quiz can be timed. |
| Step 4: Add the CheckTheAnswer() Method | Add a method to check whether the quiz taker entered a correct answer for the problem. |

| TITLE | DESCRIPTION |
|---|--|
| Step 5: Add Enter Event Handlers for the NumericUpDown Controls | Add event handlers that make your quiz easier to take. |
| Step 6: Add a Subtraction Problem | Add a subtraction problem that generates random numbers, uses the timer, and checks for correct answers. |
| Step 7: Add Multiplication and Division Problems | Add multiplication and division problems that generate random numbers, use the timer, and check for correct answers. |
| Step 8: Customize the Quiz | Try other features, such as changing colors and adding a hint. |

Step 1: Create a Project and Add Labels to Your Form

12/22/2017 • 5 min to read • [Edit Online](#)

As the first steps in developing this quiz, you create the project, and you add labels, a button, and other controls to a form. You also set properties for each control that you add. The project will contain the form, the controls, and (later in the tutorial) code. The button starts the quiz, the labels show the quiz problems, and the other controls show the quiz answers and the time that remains to finish the quiz.

NOTE

This topic is part of a tutorial series about basic coding concepts. For an overview of the tutorial, see [Tutorial 2: Create a Timed Math Quiz](#).

To create a project and and set properties for a form

1. On the menu bar, choose **File, New, Project**.
2. In the **Installed Templates** list, choose either **C#** or **Visual Basic**.
3. In the list of templates, choose the **Windows Forms Application** template, name it **Math Quiz**, and then choose the **OK** button.

A form that's named **Form1.cs** or **Form1.vb** appears, depending on the programming language that you chose.

4. Choose the form, and then change its **Text** property to **Math Quiz**.

The **Properties** window contains properties for the form.

5. Change the size of the form to 500 pixels wide by 400 pixels tall.

You can resize the form by dragging its edges until the correct size appears in the lower-left corner of the integrated development environment (IDE). As an alternative, you can change the values of the **Size** property.

6. Change the value of the **FormBorderStyle** property to **Fixed3D**, and set the **MaximizeBox** property to **False**.

These values prevent quiz takers from resizing the form.

To create the Time Remaining box

1. Add a **Label** control from the Toolbox, and then set the value of its **(Name)** property to `timeLabel`.

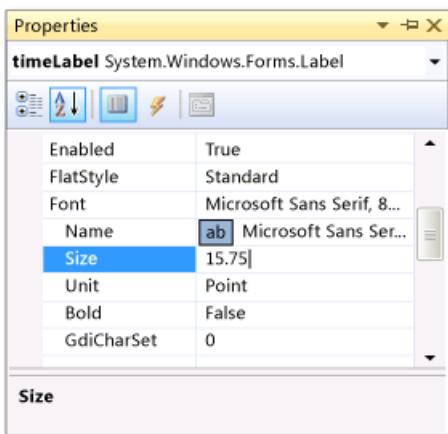
This label will become a box in the upper-right corner that shows the number of seconds that remain in the quiz.

2. Change the **AutoSize** property to **False** so that you can resize the box.
3. Change the **BorderStyle** property to **FixedSingle** to draw a line around the box.
4. Set the **Size** property to **200, 30**.
5. Move the label to the upper-right corner of the form, where blue spacer lines will appear.

These lines help you align controls on the form.

- In the **Properties** window, choose the **Text** property, and then choose the Backspace key to clear its value.
- Choose the plus sign (+) next to the **Font** property, and then change the value of the **Size** property to **15.75**.

You can change several font properties, as the following picture shows.



Properties window showing font size

- Add another **Label** control from the Toolbox, and then set its font size to **15.75**.
- Set the **Text** property to **Time Left**.
- Move the label so that it lines up just to the left of the **timeLabel** label.

To add controls for the addition problems

- Add a **Label** control from the Toolbox, and then set its **Text** property to **?** (question mark).
- Set the **AutoSize** property to **False**.
- Set the **Size** property to **60, 50**.
- Set the font size to **18**.
- Set the **.TextAlign** property to **MiddleCenter**.
- Set the **Location** property to **50, 75** to position the control on the form.
- Set the **(Name)** property to **plusLeftLabel**.
- Choose the **plusLeftLabel** label, and then choose either the **Ctrl+C** keys or **Copy** on the **Edit** menu.
- Paste the label three times by choosing either the **Ctrl+V** keys or **Paste** on the **Edit** menu.
- Arrange the three new labels so that they are in a row to the right of the **plusLeftLabel** label.

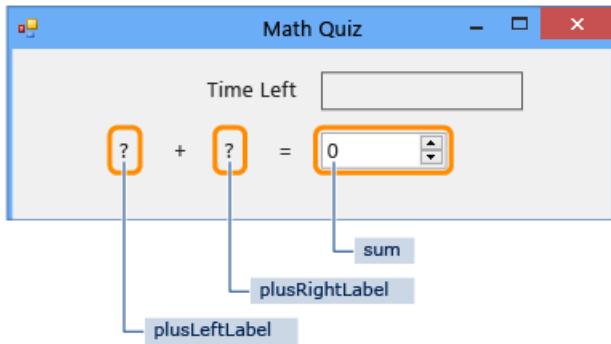
You can use the spacer lines to space them out and line them up.

- Set the value of the second label's **Text** property to **+** (plus sign).
- Set the value of the third label's **(Name)** property to **plusRightLabel**.
- Set the value of the fourth label's **Text** property to **=** (equal sign).
- Add a **NumericUpDown** control from the Toolbox, set its font size to **18**, and set its width to **100**.

You'll learn more about this kind of control later.

- Line up the **NumericUpDown** control with the label controls for the addition problem.
- Change the value of the **(Name)** property for the **NumericUpDown** control to **sum**.

You've created the first row, as the following picture shows.



First row of math quiz

To add controls for the subtraction, multiplication, and division problems

1. Copy all five controls for the addition problem (the four Label controls and the NumericUpDown control), and then paste them.

The form contains five new controls, which are still selected.

2. Move all of the controls into place so that they line up below the addition controls.

You can use the spacer lines to give enough distance between the two rows.

3. Change the value of the **Text** property for the second label to - (minus sign).
4. Name the first question-mark label **minusLeftLabel**.
5. Name the second question-mark label **minusRightLabel**.
6. Name the **NumericUpDown** control **difference**.
7. Paste the five controls two more times.
8. For the third row, name the first label **timesLeftLabel**, change the second label's **Text** property to \times (multiplication sign), name the third label **timesRightLabel**, and name the NumericUpDown control **product**.
9. For the fourth row, name the first label **dividedLeftLabel**, change the second label's **Text** property to \div (division sign), name the third label **dividedRightLabel**, and name the NumericUpDown control **quotient**.

NOTE

You can copy the multiplication sign \times and the division sign \div from this tutorial and paste them onto the form.

To add a start button and set the tab-index order

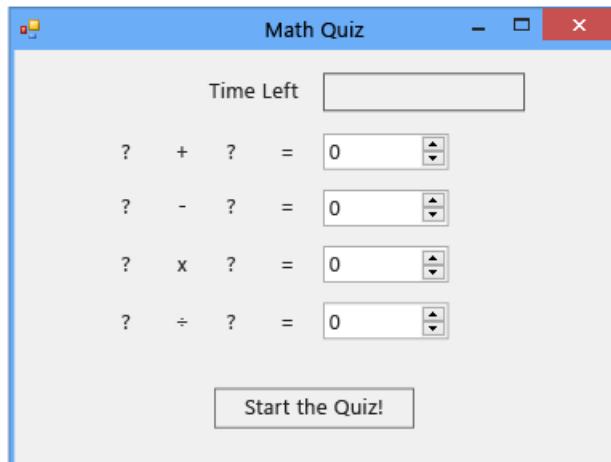
1. Add a **Button** control from the Toolbox, and then set its **(Name)** property to **startButton**.
2. Set the **Text** property to **Start the quiz**.
3. Set the font size to **14**.
4. Set the **AutoSize** property to **True**, which causes the button to automatically resize to fit the text.
5. Center the button near the bottom of the form.
6. Set the value of the **TabIndex** property for the **startButton** control to **1**.

NOTE

The **TabIndex** property sets the order of the controls when the quiz taker chooses the Tab key. To see how it works, open any dialog box (for example, on the menu bar, choose **File, Open**), and then choose the Tab key a few times. Watch how your cursor moves from control to control each time that you choose the Tab key. A programmer decided the order when creating that form.

- Set the value of the **TabIndex** property for the NumericUpDown sum control to **2**, for the difference control to **3**, for the product control to **4**, and for the quotient control to **5**.

The form should look like the following illustration.



Initial math quiz form

- To verify whether the **TabIndex** property works as you expect, save and run your program by choosing the F5 key, or by choosing **Debug, Start Debugging** on the menu bar, and then choose the Tab key a few times.

To continue or review

- To go to the next tutorial step, see [Step 2: Create a Random Addition Problem](#).
- To return to the overview topic, see [Tutorial 2: Create a Timed Math Quiz](#).

Step 2: Create a Random Addition Problem

3/1/2018 • 5 min to read • [Edit Online](#)

In the second part of this tutorial, you make the quiz challenging by adding math problems that are based on random numbers. You also create a method that's named `StartTheQuiz()` and that fills in the problems and starts the countdown timer. Later in this tutorial, you'll add the subtraction, multiplication, and division problems.

NOTE

This topic is part of a tutorial series about basic coding concepts. For an overview of the tutorial, see [Tutorial 2: Create a Timed Math Quiz](#).

To create a random addition problem

1. In the form designer, choose the form (Form1).
2. On the menu bar, choose **View, Code**.

Form1.cs or Form1.vb appears, depending on the programming language that you're using, so that you can view the code behind the form.

3. Create a `Random` object by adding a `new` statement near the top of the code, like the following.

```
public partial class Form1 : Form
{
    // Create a Random object called randomizer
    // to generate random numbers.
    Random randomizer = new Random();
```

```
Public Class Form1

    ' Create a Random object called randomizer
    ' to generate random numbers.
    Private randomizer As New Random
```

You've added a `Random` object to your form and named the object **randomizer**.

`Random` is known as an object. You've probably heard that word before, and you learn more about what it means for programming in the next tutorial. For now, just remember that you can use `new` statements to create buttons, labels, panels, OpenFileDialogs, ColorDialogs, SoundPlayers, Randoms, and even forms, and those items are referred to as objects. When you run your program, the form is started, and the code behind it creates a `Random` object and names it **randomizer**.

Soon you'll build a method to check the answers, so your quiz must use variables to store the random numbers that it generates for each problem. See [Variables](#) or [Types](#). To properly use variables, you must declare them, which means listing their names and data types.

4. Add two integer variables to the form, and name them **addend1** and **addend2**.

NOTE

An integer variable is known as an int in C# or an Integer in Visual Basic. This kind of variable stores a positive or negative number from -2147483648 through 2147483647 and can store only whole numbers, not decimals.

You use a similar syntax to add an integer variable as you did to add the `Random` object, as the following code shows.

```
// Create a Random object called randomizer
// to generate random numbers.
Random randomizer = new Random();

// These integer variables store the numbers
// for the addition problem.
int addend1;
int addend2;
```

```
' Create a Random object called randomizer
' to generate random numbers.
Private randomizer As New Random

' These integer variables store the numbers
' for the addition problem.
Private addend1 As Integer
Private addend2 As Integer
```

5. Add a method that's named `StartTheQuiz()` and that uses the `Random` object's `Next()` method to show the random numbers in the labels. `StartTheQuiz()` will eventually fill in all of the problems and then start the timer, so add a comment. The function should look like the following.

```
/// <summary>
/// Start the quiz by filling in all of the problems
/// and starting the timer.
/// </summary>
public void StartTheQuiz()
{
    // Fill in the addition problem.
    // Generate two random numbers to add.
    // Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51);
    addend2 = randomizer.Next(51);

    // Convert the two randomly generated numbers
    // into strings so that they can be displayed
    // in the label controls.
    plusLeftLabel.Text = addend1.ToString();
    plusRightLabel.Text = addend2.ToString();

    // 'sum' is the name of the NumericUpDown control.
    // This step makes sure its value is zero before
    // adding any values to it.
    sum.Value = 0;
}
```

```

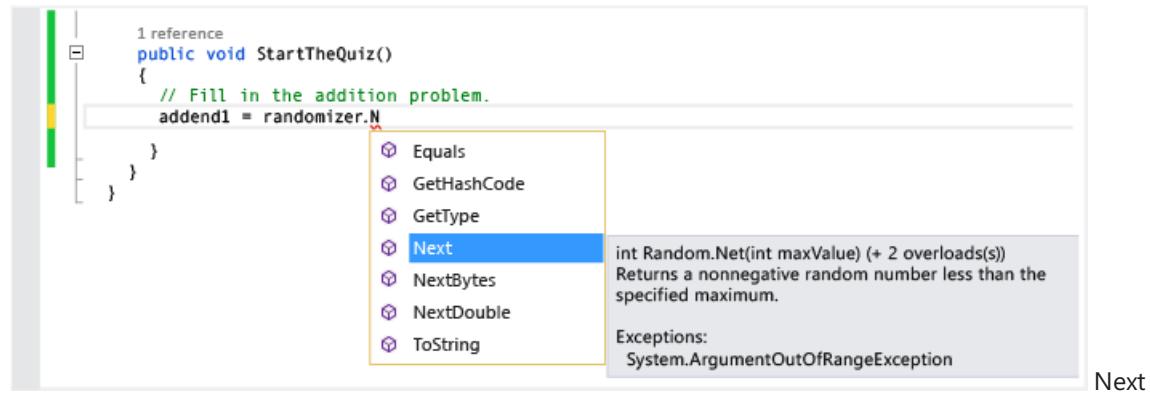
''' <summary>
''' Start the quiz by filling in all of the problems
''' and starting the timer.
''' </summary>
''' <remarks></remarks>
Public Sub StartTheQuiz()
    ' Fill in the addition problem.
    ' Generate two random numbers to add.
    ' Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51)
    addend2 = randomizer.Next(51)

    ' Convert the two randomly generated numbers
    ' into strings so that they can be displayed
    ' in the label controls.
    plusLeftLabel.Text = addend1.ToString()
    plusRightLabel.Text = addend2.ToString()

    ' 'sum' is the name of the NumericUpDown control.
    ' This step makes sure its value is zero before
    ' adding any values to it.
    sum.Value = 0
End Sub

```

Notice that when you enter the dot (.) after **randomizer** in the code, an IntelliSense window opens and shows you all of the **Random** object's methods that you can call. For example, IntelliSense lists the **Next()** method, as follows.



method

When you enter a dot after an object, IntelliSense shows a list of the object's members, such as properties, methods, and events.

NOTE

When you use the **Next()** method with the **Random** object, such as when you call **randomizer.Next(50)**, you get a random number that's less than 50 (from 0 through 49). In this example, you called **randomizer.Next(51)**. You used 51 and not 50 so that the two random numbers will add up to an answer that's from 0 through 100. If you pass 50 to the **Next()** method, it chooses a number from 0 through 49, so the highest possible answer is 98, not 100. After the first two statements in the method run, each of the two integer variables, **addend1** and **addend2**, hold a random number from 0 through 50. This screenshot shows Visual C# code, but IntelliSense works the same way for Visual Basic.

Take a closer look at these statements.

```

plusLeftLabel.Text = addend1.ToString();
plusRightLabel.Text = addend2.ToString();

```

```

' Convert the two randomly generated numbers
' into strings so that they can be displayed
' in the label controls.
plusLeftLabel.Text = addend1.ToString()
plusRightLabel.Text = addend2.ToString()

```

The statements set the **Text** properties of **plusLeftLabel** and **plusRightLabel** so that they display the two random numbers. You must use the integer's `ToString()` method to convert the numbers to text. (In programming, string means text. Label controls display only text, not numbers.)

6. In the design window, either double-click the **Start** button, or choose it and then choose the Enter key.

When a quiz taker chooses this button, the quiz should start, and you've just added a Click event handler to implement that behavior.

7. Add the following two statements.

```

private void startButton_Click(object sender, EventArgs e)
{
    StartTheQuiz();
    startButton.Enabled = false;
}

```

```

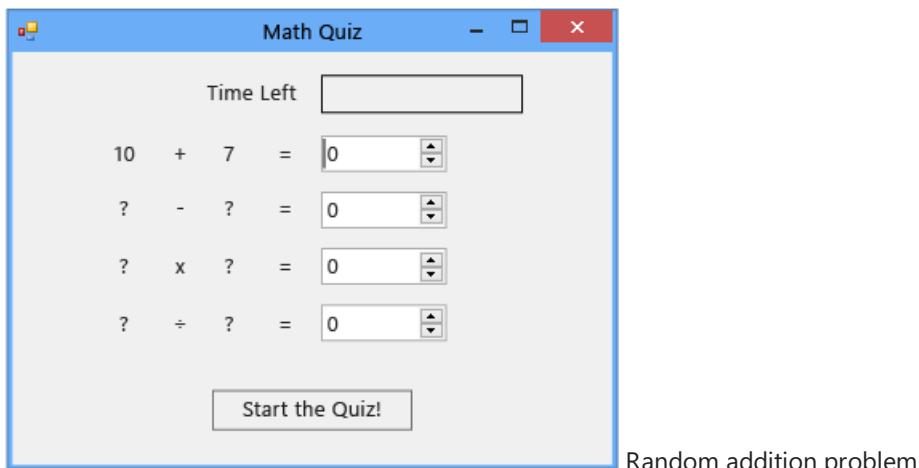
' Call the StartTheQuiz() method and enable
' the Start button.
Private Sub startButton_Click() Handles startButton.Click
    StartTheQuiz()
    startButton.Enabled = False
End Sub

```

The first statement calls the new `startTheQuiz()` method. The second statement sets the **Enabled** property of the **startButton** control to **False** so that the quiz taker can't choose the button during a quiz.

8. Save your code, run it, and then choose the **Start** button.

A random addition problem appears, as the following illustration shows.



Random addition problem

In the next step of the tutorial, you'll add the sum.

To continue or review

- To go to the next tutorial step, see [Step 3: Add a Countdown Timer](#).
- To return to the previous tutorial step, see [Step 1: Create a Project and Add Labels to Your Form](#).

Step 3: Add a Countdown Timer

12/22/2017 • 6 min to read • [Edit Online](#)

In the third part of this tutorial, you'll add a countdown timer to track the number of seconds that remain for the quiz taker to finish.

NOTE

This topic is part of a tutorial series about basic coding concepts. For an overview of the tutorial, see [Tutorial 2: Create a Timed Math Quiz](#).

To add a countdown timer

1. Add an integer variable that's named **timeLeft**, just like you did in the previous procedure. Your code should look like the following.

```
Public Class Form1

    ' Create a Random object called randomizer
    ' to generate random numbers.
    Private randomizer As New Random

    ' These integer variables store the numbers
    ' for the addition problem.
    Private addend1 As Integer
    Private addend2 As Integer

    ' This integer variable keeps track of the
    ' remaining time.
    Private timeLeft As Integer
```

```
public partial class Form1 : Form
{
    // Create a Random object called randomizer
    // to generate random numbers.
    Random randomizer = new Random();

    // These integer variables store the numbers
    // for the addition problem.
    int addend1;
    int addend2;

    // This integer variable keeps track of the
    // remaining time.
    int timeLeft;
```

Now you need a method that actually counts the seconds, such as a timer, which raises an event after the amount of time that you specify.

2. In the design window, move a **Timer** control from the **Components** category of the Toolbox to your form.

The control appears in the gray area at the bottom of the design window.

3. On the form, choose the **timer1** icon that you just added, and set its **Interval** property to **1000**.

Because the interval value is milliseconds, a value of 1000 causes the Tick event to fire every second.

4. On the form, double-click the Timer control, or choose it and then choose the Enter key.

The code editor appears and displays the method for the Tick event handler that you just added.

5. Add the following statements to the new event handler method.

```
Private Sub Timer1_Tick() Handles Timer1.Tick

    If timeLeft > 0 Then
        ' Display the new time left
        ' by updating the Time Left label.
        timeLeft -= 1
        timeLabel.Text = timeLeft & " seconds"
    Else
        ' If the user ran out of time, stop the timer, show
        ' a MessageBox, and fill in the answers.
        Timer1.Stop()
        timeLabel.Text = "Time's up!"
        MessageBox.Show("You didn't finish in time.", "Sorry!")
        sum.Value = addend1 + addend2
        startButton.Enabled = True
    End If

End Sub
```

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (timeLeft > 0)
    {
        // Display the new time left
        // by updating the Time Left label.
        timeLeft = timeLeft - 1;
        timeLabel.Text = timeLeft + " seconds";
    }
    else
    {
        // If the user ran out of time, stop the timer, show
        // a MessageBox, and fill in the answers.
        timer1.Stop();
        timeLabel.Text = "Time's up!";
        MessageBox.Show("You didn't finish in time.", "Sorry!");
        sum.Value = addend1 + addend2;
        startButton.Enabled = true;
    }
}
```

Based on what you added, the timer checks each second whether time has run out by determining whether the **timeLeft** integer variable is greater than 0. If it is, time still remains. The timer first subtracts 1 from timeLeft and then updates the **Text** property of the `timeLabel` control to show the quiz taker how many seconds remain.

If no time remains, the timer stops and changes the text of the `timeLabel` control so that it shows **Time's up!** A message box announces that the quiz is over, and the answer is revealed—in this case, by adding addend1 and addend2. The **Enabled** property of the `startButton` control is set to `true` so that the quiz taker can start another quiz.

You just added an `if else` statement, which is how you tell programs to make decisions. An `if else` statement looks like the following.

NOTE

The following example is for illustration only-don't add it to your project.

```
If (something that your program will check) Then
    ' One or more statements that will run
    ' if what the program checked is true.
Else
    ' One or more statements that will run
    ' if what the program checked is false.
End If
```

```
if (something that your program will check)
{
    // One or more statements that will run
    // if what the program checked is true.
}
else
{
    // One or more statements that will run
    // if what the program checked is false.
}
```

Look closely at the statement that you added in the `else` block to show the answer to the addition problem.

```
sum.Value = addend1 + addend2
```

```
sum.Value = addend1 + addend2;
```

The statement `addend1 + addend2` adds the values in the two variables together. The first part (`sum.Value`) uses the **Value** property of the sum `NumericUpDown` control to display the correct answer. You use the same property later to check the answers for the quiz.

Quiz takers can enter numbers more easily by using a `NumericUpDown` control, which is why you use one for the answers to the math problems. All of the potential answers are whole numbers from 0 through 100. By leaving the default values of the **Minimum**, **Maximum**, and **DecimalPlaces** properties, you ensure that quiz takers can't enter decimals, negative numbers, or numbers that are too high. (If you wanted to allow quiz takers to enter 3.141 but not 3.1415, you could set the **DecimalPlaces** property to 3.)

6. Add three lines to the end of the `startTheQuiz()` method, so the code looks like the following.

```

''' <summary>
''' Start the quiz by filling in all of the problem
''' values and starting the timer.
''' </summary>
''' <remarks></remarks>
Public Sub StartTheQuiz()

    ' Fill in the addition problem.
    ' Generate two random numbers to add.
    ' Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51)
    addend2 = randomizer.Next(51)

    ' Convert the two randomly generated numbers
    ' into strings so that they can be displayed
    ' in the label controls.
    plusLeftLabel.Text = addend1.ToString()
    plusRightLabel.Text = addend2.ToString()

    ' 'sum' is the name of the NumericUpDown control.
    ' This step makes sure its value is zero before
    ' adding any values to it.
    sum.Value = 0

    ' Start the timer.
    timeLeft = 30
    timeLabel.Text = "30 seconds"
    Timer1.Start()

End Sub

```

```

/// <summary>
/// Start the quiz by filling in all of the problem
/// values and starting the timer.
/// </summary>
public void StartTheQuiz()
{
    // Fill in the addition problem.
    // Generate two random numbers to add.
    // Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51);
    addend2 = randomizer.Next(51);

    // Convert the two randomly generated numbers
    // into strings so that they can be displayed
    // in the label controls.
    plusLeftLabel.Text = addend1.ToString();
    plusRightLabel.Text = addend2.ToString();

    // 'sum' is the name of the NumericUpDown control.
    // This step makes sure its value is zero before
    // adding any values to it.
    sum.Value = 0;

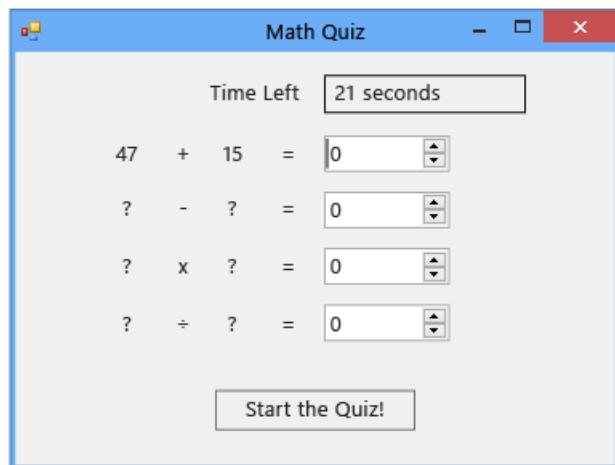
    // Start the timer.
    timeLeft = 30;
    timeLabel.Text = "30 seconds";
    timer1.Start();
}

```

Now, when your quiz starts, the **timeLeft** variable is set to 30 and the **Text** property of the **timeLabel** control is set to 30 seconds. Then the **Start()** method of the **Timer** control starts the countdown. (The quiz doesn't check the answer yet—that comes next.)

7. Save your program, run it, and then choose the **Start** button on the form.

The timer starts to count down. When time runs out, the quiz ends, and the answer appears. The following illustration shows the quiz in progress.



Math quiz in progress

To continue or review

- To go to the next tutorial step, see [Step 4: Add the CheckTheAnswer\(\) Method](#).
- To return to the previous tutorial step, see [Step 2: Create a Random Addition Problem](#).

Step 4: Add the CheckTheAnswer() Method

12/22/2017 • 3 min to read • [Edit Online](#)

In the fourth part of this tutorial, you'll write a method, `CheckTheAnswer()`, that determines whether the answers to the math problems are correct. This topic is part of a tutorial series about basic coding concepts. For an overview of the tutorial, see [Tutorial 2: Create a Timed Math Quiz](#).

NOTE

If you're following along in Visual Basic, you'll use the `Function` keyword instead of the usual `Sub` keyword because this method returns a value. It's really that simple: a sub doesn't return a value, but a function does.

To verify whether the answers are correct

1. Add the `CheckTheAnswer()` method.

When this method is called, it adds the values of `addend1` and `addend2` and compares the result to the value in the sum `NumericUpDown` control. If the values are equal, the method returns a value of `true`. Otherwise, the method returns a value of `false`. Your code should look like the following.

```
''' <summary>
''' Check the answer to see if the user got everything right.
''' </summary>
''' <returns>True if the answer's correct, false otherwise.</returns>
''' <remarks></remarks>
Public Function CheckTheAnswer() As Boolean

    If addend1 + addend2 = sum.Value Then
        Return True
    Else
        Return False
    End If

End Function
```

```
/// <summary>
/// Check the answer to see if the user got everything right.
/// </summary>
/// <returns>True if the answer's correct, false otherwise.</returns>
private bool CheckTheAnswer()
{
    if (addend1 + addend2 == sum.Value)
        return true;
    else
        return false;
}
```

Next, you'll check the answer by updating the code in the method for the timer's Tick event handler to call the new `CheckTheAnswer()` method.

2. Add the following code to the `if else` statement.

```

Private Sub Timer1_Tick() Handles Timer1.Tick

    If CheckTheAnswer() Then
        ' If CheckTheAnswer() returns true, then the user
        ' got the answer right. Stop the timer
        ' and show a MessageBox.
        Timer1.Stop()
        MessageBox.Show("You got all of the answers right!", "Congratulations!")
        startButton.Enabled = True
    ElseIf timeLeft > 0 Then
        ' If CheckTheAnswer() return false, keep counting
        ' down. Decrease the time left by one second and
        ' display the new time left by updating the
        ' Time Left label.
        timeLeft -= 1
        timeLabel.Text = timeLeft & " seconds"
    Else
        ' If the user ran out of time, stop the timer, show
        ' a MessageBox, and fill in the answers.
        Timer1.Stop()
        timeLabel.Text = "Time's up!"
        MessageBox.Show("You didn't finish in time.", "Sorry!")
        sum.Value = addend1 + addend2
        startButton.Enabled = True
    End If

End Sub

```

```

private void timer1_Tick(object sender, EventArgs e)
{
    if (CheckTheAnswer())
    {
        // If CheckTheAnswer() returns true, then the user
        // got the answer right. Stop the timer
        // and show a MessageBox.
        timer1.Stop();
        MessageBox.Show("You got all the answers right!",
                       "Congratulations!");
        startButton.Enabled = true;
    }
    else if (timeLeft > 0)
    {
        // If CheckTheAnswer() return false, keep counting
        // down. Decrease the time left by one second and
        // display the new time left by updating the
        // Time Left label.
        timeLeft--;
        timeLabel.Text = timeLeft + " seconds";
    }
    else
    {
        // If the user ran out of time, stop the timer, show
        // a MessageBox, and fill in the answers.
        timer1.Stop();
        timeLabel.Text = "Time's up!";
        MessageBox.Show("You didn't finish in time.", "Sorry!");
        sum.Value = addend1 + addend2;
        startButton.Enabled = true;
    }
}

```

If the answer is correct, `CheckTheAnswer()` returns `true`. The event handler stops the timer, shows a congratulatory message, and then makes the **Start** button available again. Otherwise, the quiz continues.

3. Save your program, run it, start a quiz, and provide a correct answer to the addition problem.

NOTE

When you enter your answer, you must either select the default value before you start to enter your answer, or you must delete the zero manually. You'll correct this behavior later in this tutorial.

When you provide a correct answer, a message box opens, the **Start** button becomes available, and the timer stops.

To continue or review

- To go to the next tutorial step, see [Step 5: Add Enter Event Handlers for the NumericUpDown Controls](#).
- To return to the previous tutorial step, see [Step 3: Add a Countdown Timer](#).

Step 5: Add Enter Event Handlers for the NumericUpDown Controls

12/22/2017 • 4 min to read • [Edit Online](#)

In the fifth part of this tutorial, you'll add Enter event handlers to make entering answers for quiz problems a little easier. This code will select and clear the current value in each NumericUpDown control as soon as the quiz taker chooses it and starts to enter a different value.

NOTE

This topic is part of a tutorial series about basic coding concepts. For an overview of the tutorial, see [Tutorial 2: Create a Timed Math Quiz](#).

To verify the default behavior

1. Run your program, and start the quiz.

In the NumericUpDown control for the addition problem, the cursor flashes next to **0** (zero).

2. Enter **3**, and note that the control shows **30**.
3. Enter **5**, and note that **350** appears but changes to **100** after a second.

Before you fix this problem, think about what's happening. Consider why the **0** didn't disappear when you entered **3** and why **350** changed to **100** but not immediately.

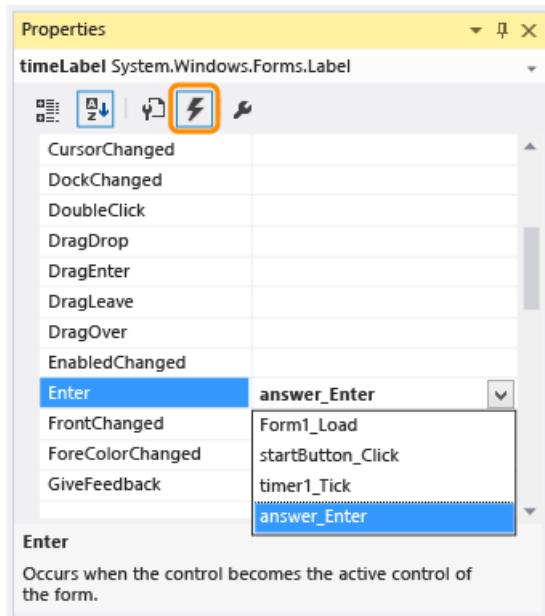
This behavior may seem odd, but it makes sense given the logic of the code. When you choose the **Start** button, its **Enabled** property is set to **False**, and the button appears dimmed and is unavailable. Your program changes the current selection (focus) to the control that has the next lowest TabIndex value, which is the NumericUpDown control for the addition problem. When you use the Tab key to go to a NumericUpDown control, the cursor is automatically positioned at the start of the control, which is why the numbers that you enter appear from the left side and not the right side. When you specify a number that's higher than the value of the **MaximumValue** property, which is set to 100, the number that you enter is replaced with the value of that property.

To add an Enter event handler for a NumericUpDown control

1. Choose the first NumericUpDown control (named "sum") on the form, and then, in the **Properties** dialog box, choose the **Events** icon on the toolbar.

The **Events** tab in the **Properties** dialog box displays all of the events that you can respond to (handle) for the item that you choose on the form. Because you chose the NumericUpDown control, all of the events listed pertain to it.

2. Choose the **Enter** event, enter `answer_Enter`, and then choose the Enter key.



Properties dialog box

You've just added an Enter event handler for the sum NumericUpDown control, and you've named the handler **answer_Enter**.

- In the method for the **answer_Enter** event handler, add the following code.

```

''' <summary>
''' Modify the behavior of the NumericUpDown control
''' to make it easier to enter numeric values for
''' the quiz.
''' </summary>
Private Sub answer_Enter(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles sum.Enter

    ' Select the whole answer in the NumericUpDown control.
    Dim answerBox = TryCast(sender, NumericUpDown)

    If answerBox IsNot Nothing Then
        Dim lengthOfAnswer = answerBox.Value.ToString().Length
        answerBox.Select(0, lengthOfAnswer)
    End If

End Sub

```

```

private void answer_Enter(object sender, EventArgs e)
{
    // Select the whole answer in the NumericUpDown control.
    NumericUpDown answerBox = sender as NumericUpDown;

    if (answerBox != null)
    {
        int lengthOfAnswer = answerBox.Value.ToString().Length;
        answerBox.Select(0, lengthOfAnswer);
    }
}

```

This code may look complex, but you can understand it if you look at it step by step. First, look at the top of the method: `object sender` in C# or `sender As System.Object` in Visual Basic. This parameter refers to the object whose event is firing, which is known as the sender. In this case, the sender object is the NumericUpDown control. So, in the first line of the method, you specify that the sender isn't just any generic object but specifically a NumericUpDown control. (Every NumericUpDown control is an object, but not every object is a NumericUpDown control.) The NumericUpDown control is named **answerBox** in this method,

because it will be used for all of the NumericUpDown controls on the form, not just the sum NumericUpDown control. Because you declare the answerBox variable in this method, its scope applies only to this method. In other words, the variable can be used only within this method.

The next line verifies whether answerBox was successfully converted (cast) from an object to a NumericUpDown control. If the conversion was unsuccessful, the variable would have a value of `null` (C#) or `Nothing` (Visual Basic). The third line gets the length of the answer that appears in the NumericUpDown control, and the fourth line selects the current value in the control based on this length. Now, when the quiz taker chooses the control, Visual Studio fires this event, which causes the current answer to be selected. As soon as the quiz taker starts to enter a different answer, the previous answer is cleared and replaced with the new answer.

4. In Windows Forms Designer, choose the difference NumericUpDown control.
5. In the **Events** page of the **Properties** dialog box, scroll down to the **Enter** event, choose the drop-down arrow at the end of the row, and then choose the `answer_Enter` event handler that you just added.
6. Repeat the previous step for the product and quotient NumericUpDown controls.
7. Save your program, and then run it.

When you choose a NumericUpDown control, the existing value is automatically selected and then cleared when you start to enter a different value.

To continue or review

- To go to the next tutorial step, see [Step 6: Add a Subtraction Problem](#).
- To return to the previous tutorial step, see [Step 4: Add the CheckTheAnswer\(\) Method](#).

Step 6: Add a Subtraction Problem

3/1/2018 • 6 min to read • [Edit Online](#)

In the sixth part of this tutorial, you'll add a subtraction problem and learn how to perform the following tasks:

- Store the subtraction values.
- Generate random numbers for the problem (and be sure that the answer is between 0 and 100).
- Update the method that checks the answers so that it checks the new subtraction problem too.
- Update your timer's Tick event handler so that the event handler fills in the correct answer when time runs out.

To add a subtraction problem

1. Add two integer variables for the subtraction problem to your form, between the integer variables for the addition problem and the timer. The code should look like the following.

```
Public Class Form1

    ' Create a Random object called randomizer
    ' to generate random numbers.
    Private randomizer As New Random

    ' These integer variables store the numbers
    ' for the addition problem.
    Private addend1 As Integer
    Private addend2 As Integer

    ' These integer variables store the numbers
    ' for the subtraction problem.
    Private minuend As Integer
    Private subtrahend As Integer

    ' This integer variable keeps track of the
    ' remaining time.
    Private timeLeft As Integer
```

```
public partial class Form1 : Form
{
    // Create a Random object called randomizer
    // to generate random numbers.
    Random randomizer = new Random();

    // These integer variables store the numbers
    // for the addition problem.
    int addend1;
    int addend2;

    // These integer variables store the numbers
    // for the subtraction problem.
    int minuend;
    int subtrahend;

    // This integer variable keeps track of the
    // remaining time.
    int timeLeft;
```

The names of the new integer variables—**minuend** and **subtrahend**—aren't programming terms. They're the traditional names in arithmetic for the number that's being subtracted (the subtrahend) and the number from which the subtrahend is being subtracted (the minuend). The difference is the minuend minus the subtrahend. You could use other names, because your program doesn't require specific names for variables, controls, components, or methods. You must follow rules such as not starting names with digits, but you can generally use names such as `x1`, `x2`, `x3`, and `x4`. However, generic names make code difficult to read and problems nearly impossible to track down. To keep variable names unique and helpful, you'll use the traditional names for multiplication (`multiplicand × multiplier = product`) and division (`dividend ÷ divisor = quotient`) later in this tutorial.

Next, you'll modify the `StartTheQuiz()` method to provide random values for the subtraction problem.

2. Add the following code after the "Fill in the subtraction problem" comment.

```
''' <summary>
''' Start the quiz by filling in all of the problem
''' values and starting the timer.
''' </summary>
''' <remarks></remarks>
Public Sub StartTheQuiz()

    ' Fill in the addition problem.
    ' Generate two random numbers to add.
    ' Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51)
    addend2 = randomizer.Next(51)

    ' Convert the two randomly generated numbers
    ' into strings so that they can be displayed
    ' in the label controls.
    plusLeftLabel.Text = addend1.ToString()
    plusRightLabel.Text = addend2.ToString()

    ' 'sum' is the name of the NumericUpDown control.
    ' This step makes sure its value is zero before
    ' adding any values to it.
    sum.Value = 0

    ' Fill in the subtraction problem.
    minuend = randomizer.Next(1, 101)
    subtrahend = randomizer.Next(1, minuend)
    minusLeftLabel.Text = minuend.ToString()
    minusRightLabel.Text = subtrahend.ToString()
    difference.Value = 0

    ' Start the timer.
    timeLeft = 30
    timeLabel.Text = "30 seconds"
    Timer1.Start()

End Sub
```

```

/// <summary>
/// Start the quiz by filling in all of the problem
/// values and starting the timer.
/// </summary>
public void StartTheQuiz()
{
    // Fill in the addition problem.
    // Generate two random numbers to add.
    // Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51);
    addend2 = randomizer.Next(51);

    // Convert the two randomly generated numbers
    // into strings so that they can be displayed
    // in the label controls.
    plusLeftLabel.Text = addend1.ToString();
    plusRightLabel.Text = addend2.ToString();

    // 'sum' is the name of the NumericUpDown control.
    // This step makes sure its value is zero before
    // adding any values to it.
    sum.Value = 0;

    // Fill in the subtraction problem.
    minuend = randomizer.Next(1, 101);
    subtrahend = randomizer.Next(1, minuend);
    minusLeftLabel.Text = minuend.ToString();
    minusRightLabel.Text = subtrahend.ToString();
    difference.Value = 0;

    // Start the timer.
    timeLeft = 30;
    timeLabel.Text = "30 seconds";
    timer1.Start();
}

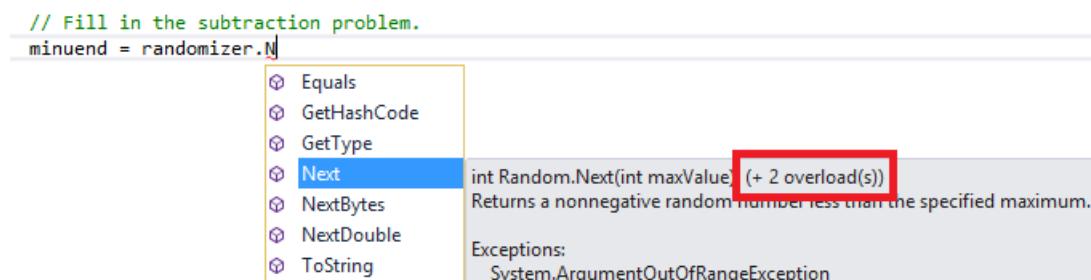
```

To prevent negative answers for the subtraction problem, this code uses the `Next()` method of the `Random` class a little differently from how the addition problem does. When you give the `Next()` method two values, it picks a random number that's greater than or equal to the first value and less than the second one. The following code chooses a random number from 1 through 100 and stores it in the `minuend` variable.

```
minuend = randomizer.Next(1, 101)
```

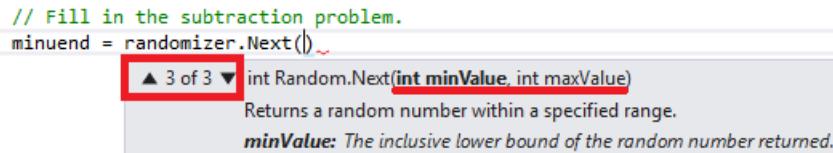
```
minuend = randomizer.Next(1, 101);
```

You can call the `Next()` method of the `Random` class, which you named "randomizer" earlier in this tutorial, in multiple ways. Methods that you can call in more than one way are referred to as overloaded, and you can use IntelliSense to explore them. Look again at the tooltip of the IntelliSense window for the `Next()` method.



IntelliSense window tooltip

The tooltip shows (**+ 2 overload(s)**), which means that you can call the `Next()` method in two other ways. Overloads contain different numbers or types of arguments, so that they work slightly differently from one another. For example, a method might take a single integer argument, and one of its overloads might take an integer and a string. You choose the correct overload based on what you want it to do. When you add the code to the `startTheQuiz()` method, more information appears in the IntelliSense window as soon as you enter `randomizer.Next()`. To cycle through the overloads, choose the Up Arrow and Down Arrow keys as shown in the following illustration:



Overload for `Next()` method

in IntelliSense

In this case, you want to choose the last overload, because you can specify minimum and maximum values.

3. Modify the `CheckTheAnswer()` method to check for the correct subtraction answer.

```
''' <summary>
''' Check the answers to see if the user got everything right.
''' </summary>
''' <returns>True if the answer's correct, false otherwise.</returns>
''' <remarks></remarks>
Public Function CheckTheAnswer() As Boolean

    If addend1 + addend2 = sum.Value AndAlso
        minuend - subtrahend = difference.Value Then

        Return True
    Else
        Return False
    End If

End Function
```

```
/// <summary>
/// Check the answers to see if the user got everything right.
/// </summary>
/// <returns>True if the answer's correct, false otherwise.</returns>
private bool CheckTheAnswer()
{
    if ((addend1 + addend2 == sum.Value)
        && (minuend - subtrahend == difference.Value))
        return true;
    else
        return false;
}
```

In Visual C#, `&&` is the `logical and` operator. In Visual Basic, the equivalent operator is `AndAlso`. These operators indicate "If the sum of `addend1` and `addend2` equals the value of the `sum` `NumericUpDown` and if `minuend` minus `subtrahend` equals the value of the `difference` `NumericUpDown`." The `CheckTheAnswer()` method returns `true` only if the answers to the addition and the subtraction problems are both correct.

4. Replace the last part of the timer's Tick event handler with the following code so that it fills in the correct answer when time runs out.

```

Else
    ' If the user ran out of time, stop the timer, show
    ' a MessageBox, and fill in the answers.
    Timer1.Stop()
    timeLabel.Text = "Time's up!"
    MessageBox.Show("You didn't finish in time.", "Sorry!")
    sum.Value = addend1 + addend2
    difference.Value = minuend - subtrahend
    startButton.Enabled = True
End If

```

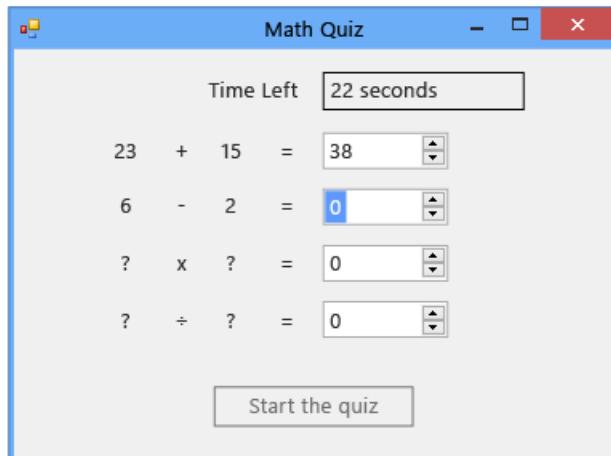
```

else
{
    // If the user ran out of time, stop the timer, show
    // a MessageBox, and fill in the answers.
    timer1.Stop();
    timeLabel.Text = "Time's up!";
    MessageBox.Show("You didn't finish in time.", "Sorry!");
    sum.Value = addend1 + addend2;
    difference.Value = minuend - subtrahend;
    startButton.Enabled = true;
}

```

5. Save and run your code.

Your program includes a subtraction problem, as the following illustration shows:



Math quiz with subtraction problem

To continue or review

- To go to the next tutorial step, see [Step 7: Add Multiplication and Division Problems](#).
- To return to the previous tutorial step, see [Step 5: Add Enter Event Handlers for the NumericUpDown Controls](#).

Step 7: Add Multiplication and Division Problems

12/22/2017 • 5 min to read • [Edit Online](#)

In the seventh part of this tutorial, you'll add multiplication and division problems, but first think about how to make that change. Consider the initial step, which involves storing values.

To add multiplication and division problems

1. Add four more integer variables to the form.

```
Public Class Form1

    ' Create a Random object called randomizer
    ' to generate random numbers.
    Private randomizer As New Random

    ' These integer variables store the numbers
    ' for the addition problem.
    Private addend1 As Integer
    Private addend2 As Integer

    ' These integer variables store the numbers
    ' for the subtraction problem.
    Private minuend As Integer
    Private subtrahend As Integer

    ' These integer variables store the numbers
    ' for the multiplication problem.
    Private multiplicand As Integer
    Private multiplier As Integer

    ' These integer variables store the numbers
    ' for the division problem.
    Private dividend As Integer
    Private divisor As Integer

    ' This integer variable keeps track of the
    ' remaining time.
    Private timeLeft As Integer
```

```
public partial class Form1 : Form
{
    // Create a Random object called randomizer
    // to generate random numbers.
    Random randomizer = new Random();

    // These integer variables store the numbers
    // for the addition problem.
    int addend1;
    int addend2;

    // These integer variables store the numbers
    // for the subtraction problem.
    int minuend;
    int subtrahend;

    // These integer variables store the numbers
    // for the multiplication problem.
    int multiplicand;
    int multiplier;

    // These integer variables store the numbers
    // for the division problem.
    int dividend;
    int divisor;

    // This integer variable keeps track of the
    // remaining time.
    int timeLeft;
```

2. As you did before, modify the `StartTheQuiz()` method to fill in random numbers for the multiplication and division problems.

```

''' <summary>
''' Start the quiz by filling in all of the problem
''' values and starting the timer.
''' </summary>
''' <remarks></remarks>
Public Sub StartTheQuiz()

    ' Fill in the addition problem.
    ' Generate two random numbers to add.
    ' Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51)
    addend2 = randomizer.Next(51)

    ' Convert the two randomly generated numbers
    ' into strings so that they can be displayed
    ' in the label controls.
    plusLeftLabel.Text = addend1.ToString()
    plusRightLabel.Text = addend2.ToString()

    ' 'sum' is the name of the NumericUpDown control.
    ' This step makes sure its value is zero before
    ' adding any values to it.
    sum.Value = 0

    ' Fill in the subtraction problem.
    minuend = randomizer.Next(1, 101)
    subtrahend = randomizer.Next(1, minuend)
    minusLeftLabel.Text = minuend.ToString()
    minusRightLabel.Text = subtrahend.ToString()
    difference.Value = 0

    ' Fill in the multiplication problem.
    multiplicand = randomizer.Next(2, 11)
    multiplier = randomizer.Next(2, 11)
    timesLeftLabel.Text = multiplicand.ToString()
    timesRightLabel.Text = multiplier.ToString()
    product.Value = 0

    ' Fill in the division problem.
    divisor = randomizer.Next(2, 11)
    Dim temporaryQuotient As Integer = randomizer.Next(2, 11)
    dividend = divisor * temporaryQuotient
    dividedLeftLabel.Text = dividend.ToString()
    dividedRightLabel.Text = divisor.ToString()
    quotient.Value = 0

    ' Start the timer.
    timeLeft = 30
    timeLabel.Text = "30 seconds"
    Timer1.Start()

End Sub

```

```

/// <summary>
/// Start the quiz by filling in all of the problem
/// values and starting the timer.
/// </summary>
public void StartTheQuiz()
{
    // Fill in the addition problem.
    // Generate two random numbers to add.
    // Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51);
    addend2 = randomizer.Next(51);

    // Convert the two randomly generated numbers
    // into strings so that they can be displayed
    // in the label controls.
    plusLeftLabel.Text = addend1.ToString();
    plusRightLabel.Text = addend2.ToString();

    // 'sum' is the name of the NumericUpDown control.
    // This step makes sure its value is zero before
    // adding any values to it.
    sum.Value = 0;

    // Fill in the subtraction problem.
    minuend = randomizer.Next(1, 101);
    subtrahend = randomizer.Next(1, minuend);
    minusLeftLabel.Text = minuend.ToString();
    minusRightLabel.Text = subtrahend.ToString();
    difference.Value = 0;

    // Fill in the multiplication problem.
    multiplicand = randomizer.Next(2, 11);
    multiplier = randomizer.Next(2, 11);
    timesLeftLabel.Text = multiplicand.ToString();
    timesRightLabel.Text = multiplier.ToString();
    product.Value = 0;

    // Fill in the division problem.
    divisor = randomizer.Next(2, 11);
    int temporaryQuotient = randomizer.Next(2, 11);
    dividend = divisor * temporaryQuotient;
    dividedLeftLabel.Text = dividend.ToString();
    dividedRightLabel.Text = divisor.ToString();
    quotient.Value = 0;

    // Start the timer.
    timeLeft = 30;
    timeLabel.Text = "30 seconds";
    timer1.Start();
}

```

3. Modify the `CheckTheAnswer()` method so that it also checks the multiplication and division problems.

```

''' <summary>
''' Check the answers to see if the user got everything right.
''' </summary>
''' <returns>True if the answer's correct, false otherwise.</returns>
''' <remarks></remarks>
Public Function CheckTheAnswer() As Boolean

    If addend1 + addend2 = sum.Value AndAlso
        minuend - subtrahend = difference.Value AndAlso
        multiplicand * multiplier = product.Value AndAlso
        dividend / divisor = quotient.Value Then

            Return True
        Else
            Return False
        End If

    End Function

```

```

/// <summary>
/// Check the answers to see if the user got everything right.
/// </summary>
/// <returns>True if the answer's correct, false otherwise.</returns>
private bool CheckTheAnswer()
{
    if ((addend1 + addend2 == sum.Value)
        && (minuend - subtrahend == difference.Value)
        && (multiplicand * multiplier == product.Value)
        && (dividend / divisor == quotient.Value))
        return true;
    else
        return false;
}

```

You can't easily enter the multiplication sign (\times) and the division sign (\div) using the keyboard, so Visual C# and Visual Basic accept an asterisk (*) for multiplication and a slash mark (/) for division.

4. Change the last part of the timer's Tick event handler so that it fills in the correct answer when time runs out.

```

Else
    ' If the user ran out of time, stop the timer, show
    ' a MessageBox, and fill in the answers.
    Timer1.Stop()
    timeLabel.Text = "Time's up!"
    MessageBox.Show("You didn't finish in time.", "Sorry!")
    sum.Value = addend1 + addend2
    difference.Value = minuend - subtrahend
    product.Value = multiplicand * multiplier
    quotient.Value = dividend / divisor
    startButton.Enabled = True
End If

```

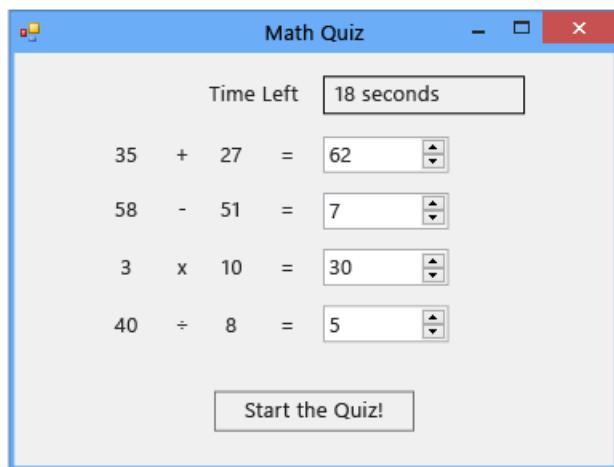
```

else
{
    // If the user ran out of time, stop the timer, show
    // a MessageBox, and fill in the answers.
    timer1.Stop();
    timeLabel.Text = "Time's up!";
    MessageBox.Show("You didn't finish in time.", "Sorry");
    sum.Value = addend1 + addend2;
    difference.Value = minuend - subtrahend;
    product.Value = multiplicand * multiplier;
    quotient.Value = dividend / divisor;
    startButton.Enabled = true;
}

```

5. Save and run your program.

Quiz takers must answer four problems to complete the quiz, as the following illustration shows.



Math quiz with four problems

To continue or review

- To go to the next tutorial step, see [Step 8: Customize the Quiz](#).
- To return to the previous tutorial step, see [Step 6: Add a Subtraction Problem](#).

Step 8: Customize the Quiz

12/22/2017 • 1 min to read • [Edit Online](#)

In the last part of the tutorial, you'll explore some ways to customize the quiz and expand on what you've already learned. For example, think about how the program creates random division problems for which the answer is never a fraction. To learn more, turn the `timeLabel` control a different color, and give the quiz taker a hint.

To customize the quiz

- When only five seconds remain in a quiz, turn the **timeLabel** control red by setting its **BackColor** property (`timeLabel.BackColor = Color.Red;`). Reset the color when the quiz is over.
- Give the quiz taker a hint by playing a sound when the correct answer is entered into a **NumericUpDown** control. (You must write an event handler for each control's `ValueChanged()` event, which fires whenever the quiz taker changes the control's value.)

To continue or review

- To download a completed version of the quiz, see [Complete Math Quiz tutorial sample](#).
- To go to the next tutorial, see [Tutorial 3: Create a Matching Game](#).
- To return to the previous tutorial step, see [Step 7: Add Multiplication and Division Problems](#).

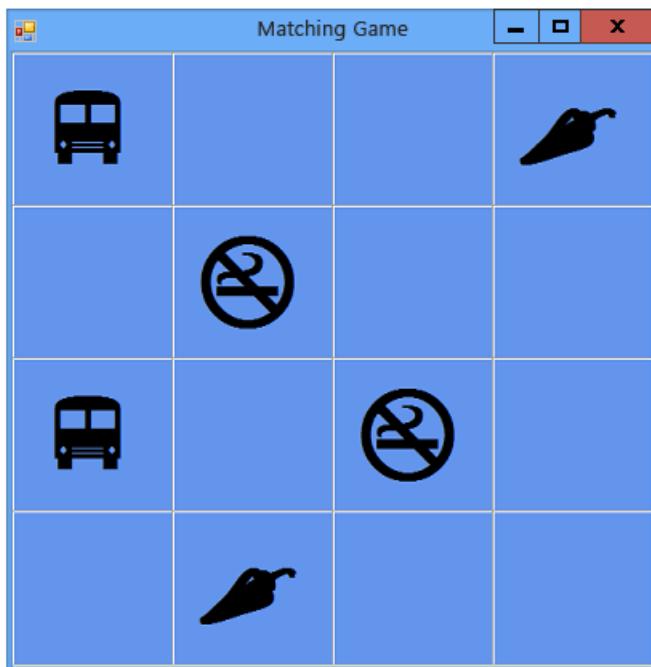
Tutorial 3: Create a Matching Game

12/22/2017 • 2 min to read • [Edit Online](#)

In this tutorial, you build a matching game, where the player must match pairs of hidden icons. You learn how to:

- Store objects, such as icons, in a `List` object.
- Use a `foreach` loop in Visual C# or a `For Each` loop in Visual Basic to iterate through items in a list.
- Keep track of a form's state by using reference variables.
- Build an event handler to respond to events that you can use with multiple objects.
- Make a timer that counts down and then fires an event exactly once after being started.

When you finish this tutorial, your program will look like the following picture.



Game that you create in this tutorial

To download a completed version of the sample, see [Complete Matching Game tutorial sample](#).

NOTE

In this tutorial, both Visual C# and Visual Basic are covered, so focus on information specific to the programming language that you're using.

If you get stuck or have programming questions, try posting your question on one of the MSDN forums. See [Visual Basic Forum](#) and [Visual C# Forum](#). Also, there are great, free video learning resources available to you. To learn more about programming in Visual Basic, see [Visual Basic Fundamentals: Development for Absolute Beginners](#). To learn more about programming in Visual C#, see [C# Fundamentals: Development for Absolute Beginners](#).

Related Topics

| TITLE | DESCRIPTION |
|---|---|
| Step 1: Create a Project and Add a Table to Your Form | Begin by creating the project and adding a <code>TableLayoutPanel</code> control to keep the controls aligned properly. |
| Step 2: Add a Random Object and a List of Icons | Add a <code>Random</code> object and a <code>List</code> object, to create a list of icons. |
| Step 3: Assign a Random Icon to Each Label | Assign the icons randomly to the <code>Label</code> controls, so that each game is different. |
| Step 4: Add a Click Event Handler to Each Label | Add a Click event handler that changes the color of the label that is clicked. |
| Step 5: Add Label References | Add reference variables to keep track of which labels are clicked. |
| Step 6: Add a Timer | Add a timer to the form to keep track of the time that has passed in the game. |
| Step 7: Keep Pairs Visible | Keep pairs of icons visible, if a matching pair is selected. |
| Step 8: Add a Method to Verify Whether the Player Won | Add a <code>CheckForWinner()</code> method to verify whether the player won. |
| Step 9: Try Other Features | Try other features, such as changing icons and colors, adding a grid, and adding sounds. Try making the board bigger and adjusting the timer. |

Step 1: Create a Project and Add a Table to Your Form

12/22/2017 • 4 min to read • [Edit Online](#)

The first step in creating a matching game is to create the project and add a table to your form. The table helps align the icons into an orderly 4x4 grid. You also set several properties to enhance the appearance of the game board.

To create a project and add a table to your form

1. On the menu bar, choose **File, New, Project**.
2. If you're not using Visual Studio Express, you need to select a programming language first. From the **Installed Templates** list, choose either **Visual C#** or **Visual Basic**.
3. In the list of project templates, choose **Windows Forms Application**, name the project **MatchingGame**, and then choose the **OK** button.
4. In the **Properties** window, set the following form properties.
 - a. Change the form's **Text** property from **Form1** to **Matching Game**. This text appears at the top of the game window.
 - b. Set the size of the form to 550 pixels wide by 550 tall. You can do this either by setting the **Size** property to **550, 550**, or by dragging the corner of the form until you see the correct size in the lower-right corner of the integrated development environment (IDE).
5. Display the toolbox by choosing the **Toolbox** tab on the left side of the IDE.
6. Drag a **TableLayoutPanel** control from the **Containers** category in the toolbox, and then set the following properties for it.
 - a. Set the **BackColor** property to **CornflowerBlue**. To do this, open the **BackColor** dialog box by choosing the drop-down arrow next to the **BackColor** property in the **Properties** window. Then, choose the **Web** tab in the **BackColor** dialog box to view a list of available color names.

NOTE

The colors are not in alphabetical order, and CornflowerBlue is near the bottom of the list.

- b. Set the **Dock** property to **Fill** by choosing the drop-down button next to the property and choosing the large middle button. This spreads the table out so that it covers the entire form.
- c. Set the **CellBorderStyle** property to **Inset**. This provides visual borders between each cell on the board.
- d. Choose the triangle button in the upper-right corner of the TableLayoutPanel to display its task menu.
- e. On the task menu, choose **Add Row** twice to add two more rows, and then choose **Add Column** twice to add two more columns.
- f. On the task menu, choose **Edit Rows and Columns** to open the **Column and Row Styles** window. Choose each of the columns, choose the **Percent** option button, and then set each column's width to 25 percent of the total width. Then select **Rows** from the drop-down box at the top of the window,

and set each row's height to 25 percent. When you're done, choose the **OK** button.

Your TableLayoutPanel should now be a 4x4 grid, with sixteen equally sized square cells. These rows and columns are where the icon images will appear later.

7. Be certain that the TableLayoutPanel is selected in the form editor. To verify this, you should see **tableLayoutPanel1** at the top of the **Properties** window. If it is not selected, choose the TableLayoutPanel on the form, or choose it in the dropdown control at the top of the **Properties** window.

While the TableLayoutPanel is selected, open the toolbox and add a **Label** control (located in the **Common Controls** category) to the upper-left cell of the TableLayoutPanel. The **Label1** control should now be selected in the IDE. Set the following properties for it.

- a. Be sure that the label's **BackColor** property is set to **CornflowerBlue**.
- b. Set the **AutoSize** property to **False**.
- c. Set the **Dock** property to **Fill**.
- d. Set the **TextAlign** property to **MiddleCenter** by choosing the drop-down button next to the property, and then choosing the middle button. This ensures the icon appears in the middle of the cell.
- e. Choose the **Font** property. An ellipsis (...) button should appear.
- f. Choose the ellipsis button, and set the **Font** value to **Webdings**, the **Font Style** to **Bold**, and the **Size** to **72**.
- g. Set the **Text** property of the label to the letter **c**.

The upper-left cell in the TableLayoutPanel should now contain a black box centered on a blue background.

NOTE

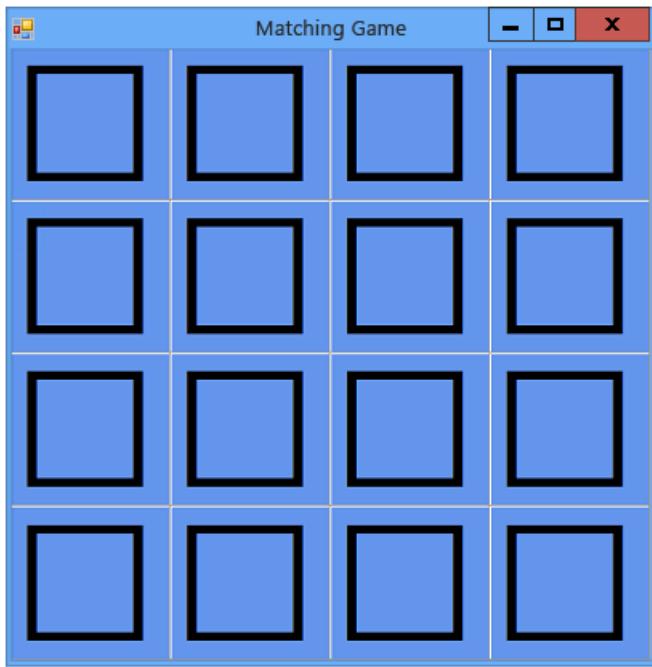
The Webdings font is a font of icons that ships with the Windows operating system. In your matching game, the player needs to match pairs of icons, so you use this font to display the icons to match. Instead of putting **c** in the **Text** property, try entering different letters to see what icons are displayed. An exclamation point is a spider, an uppercase N is an eye, and a comma is a chili pepper.

8. Choose your label control and copy it to the next cell in the TableLayoutPanel. (Choose the **Ctrl+C** keys, or on the menu bar, choose **Edit, Copy**.) Then paste it. (Choose the **Ctrl+V** keys, or on the menu bar, choose **Edit, Paste**.) A copy of the first label appears in the second cell of the TableLayoutPanel. Paste it again, and another label appears in the third cell. Keep pasting **Label** controls until all of the cells are filled.

NOTE

If you paste too many times, the IDE adds a new row to the TableLayoutPanel so that it has a place to add your new label control. You can undo it. To remove the new cell, choose the **Ctrl+Z** keys, or on the menu bar, choose **Edit, Undo**.

Now your form is laid out. It should look like the following picture.



Initial matching game form

To continue or review

- To go to the next tutorial step, see [Step 2: Add a Random Object and a List of Icons](#).
- To return to the overview topic, see [Tutorial 3: Create a Matching Game](#).

Step 2: Add a Random Object and a List of Icons

3/1/2018 • 4 min to read • [Edit Online](#)

In this step, you create a set of matching symbols for the game. Each symbol is added to two random cells in the TableLayoutPanel on the form. To do this, you use two `new` statements to create two objects. The first is a `Random` object, like the one you used in the math quiz game. It is used in this code to randomly choose cells in the TableLayoutPanel. The second object, which may be new to you, is a `List` object which is used to store the randomly-chosen symbols.

To add a Random object and a list of icons

1. In **Solution Explorer**, choose **Form1.cs** if you're using Visual C#, or **Form1.vb** if you're using Visual Basic, and then on the menu bar, choose **View, Code**. As an alternative, you can choose the **F7** key or double-click **Form1** in **Solution Explorer**.

This displays the code module behind Form1.

2. In the existing code, add the following code.

```
public partial class Form1 : Form
{
    // Use this Random object to choose random icons for the squares
    Random random = new Random();

    // Each of these letters is an interesting icon
    // in the Webdings font,
    // and each icon appears twice in this list
    List<string> icons = new List<string>()
    {
        "!", "!", "N", "N", ",", ",", "k", "k",
        "b", "b", "v", "v", "w", "w", "z", "z"
    };
}
```

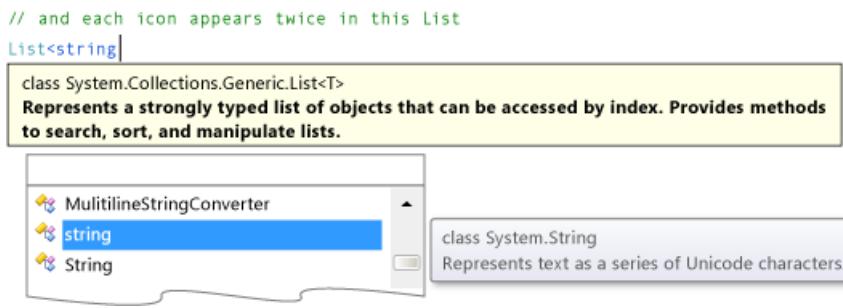
```
Public Class Form1

    ' Use this Random object to choose random icons for the squares
    Private random As New Random

    ' Each of these letters is an interesting icon
    ' in the Webdings font,
    ' and each icon appears twice in this list
    Private icons =
        New List(Of String) From {"!", "!", "N", "N", ",", ",", "k", "k",
                                "b", "b", "v", "v", "w", "w", "z", "z"}
```

If you're using Visual C#, be sure you put the code after the opening curly brace and just after the class declaration (`public partial class Form1 : Form`). If you're using Visual Basic, put the code right after the class declaration (`Public Class Form1`).

3. When adding the `List` object, notice the **IntelliSense** window that opens. The following is a Visual C# example, but similar text appears when you add a list in Visual Basic.



IntelliSense window

NOTE

The IntelliSense window appears only when you enter code manually. If you copy and paste the code, it doesn't appear.

If you look at the code (and remarks) in small sections, it's easier to understand. Your programs can use `List` objects to keep track of many different types of items. A list can hold numbers, true/false values, text, or other objects. You can even have a `List` object that holds other `List` objects. The items in a list are called *elements*, and each list only holds one type of element. So, a list of numbers can only hold numbers—you can't add text to that list. Similarly, you can't add numbers to a list of true/false values.

When you create a `List` object using a `new` statement, you need to specify the kind of data you want to store in it. That's why the tooltip at the top of the **IntelliSense** window shows the types of elements in the list. Also, that's what `List<string>` (in Visual C#) and `List(Of String)` (in Visual Basic) means: It's a `List` object that holds elements of `string` data type. A string is what your program uses to store text, which is what the tooltip to the right of the **IntelliSense** window is telling you.

4. Consider why in Visual Basic a temporary array must be created first, but in Visual C#, the list can be created with one statement. This is because the Visual C# language has *collection initializers*, which prepare the list to accept values. In Visual Basic, you can use a collection initializer. However, for compatibility with the previous version of Visual Basic, we recommend using the preceding code.

When you use a collection initializer with a `new` statement, after the new `List` object is created, the program fills it with the data you provided inside the curly braces. In this case, you get a list of strings named **icons**, and that list will be initialized so that it contains sixteen strings. Each of those strings is a single letter, and they all correspond to the icons that will be in the labels. So, the game will have a pair of exclamation points, a pair of uppercase N letters, a pair of commas, and so on. (When these characters are set to the Webdings font, they will appear as symbols, such as a bus, a bike, a spider, and so forth.) Your `List` object will have sixteen strings in all, one for each cell in the `TableLayoutPanel` panel.

NOTE

In Visual Basic, you get the same result, but first the strings are put into a temporary array, which is then converted into a `List` object. An array is similar to a list, except, for example, arrays are created with a fixed size. Lists can shrink and grow as needed, which is important in this program.

To continue or review

- To go to the next tutorial step, see [Step 3: Assign a Random Icon to Each Label](#).
- To return to the previous tutorial step, see [Step 1: Create a Project and Add a Table to Your Form](#).

Step 3: Assign a Random Icon to Each Label

12/22/2017 • 6 min to read • [Edit Online](#)

If the icons show up in the same cells every game, it's not very challenging. To avoid this, assign the icons randomly to the label controls on your form by using an `AssignIconsToSquares()` method.

To assign a random icon to each label

1. Before adding the following code, consider how the method works. There's a new keyword: `foreach` in Visual C# and `For Each` in Visual Basic. (One of the lines is commented out on purpose, which is explained at the end of this procedure.)

```
/// <summary>
/// Assign each icon from the list of icons to a random square
/// </summary>
private void AssignIconsToSquares()
{
    // The TableLayoutPanel has 16 labels,
    // and the icon list has 16 icons,
    // so an icon is pulled at random from the list
    // and added to each label
    foreach (Control control in tableLayoutPanel1.Controls)
    {
        Label iconLabel = control as Label;
        if (iconLabel != null)
        {
            int randomNumber = random.Next(Icons.Count);
            iconLabel.Text = Icons[randomNumber];
            // iconLabel.ForeColor = iconLabel.BackColor;
            Icons.RemoveAt(randomNumber);
        }
    }
}
```

```
''' <summary>
''' Assign each icon from the list of icons to a random square
''' </summary>
''' <remarks></remarks>
Private Sub AssignIconsToSquares()

    ' The TableLayoutPanel has 16 labels,
    ' and the icon list has 16 icons,
    ' so an icon is pulled at random from the list
    ' and added to each label
    For Each control In TableLayoutPanel1.Controls
        Dim iconLabel = TryCast(control, Label)
        If iconLabel IsNot Nothing Then
            Dim randomNumber = random.Next(Icons.Count)
            iconLabel.Text = Icons(randomNumber)
            ' iconLabel.ForeColor = iconLabel.BackColor
            Icons.RemoveAt(randomNumber)
        End If
    Next

End Sub
```

2. Add the `AssignIconsToSquares()` method as shown in the previous step. You can put it just below the code you added in [Step 2: Add a Random Object and a List of Icons](#).

As mentioned earlier, there's something new in your `AssignIconsToSquares()` method: a `foreach` loop in Visual C# and `For Each` in Visual Basic. You can use a `For Each` loop any time you want to do the same action multiple times. In this case, you want to execute the same statements for every label on your `TableLayoutPanel`, as explained by the following code. The first line creates a variable named `control` that stores each control one at a time while that control has the statements in the loop executed on it.

```
foreach (Control control in tableLayoutPanel1.Controls)
{
    // The statements you want to execute
    // for each label go here
    // The statements use iconLabel to access
    // each label's properties and methods
}
```

```
For Each control In TableLayoutPanel1.Controls
    ' The statements you want to execute
    ' for each label go here
    ' The statements use iconLabel to access
    ' each label's properties and methods
Next
```

NOTE

The names "iconLabel" and "control" are used because they are descriptive. You can replace these names with any names, and the code will work exactly the same as long as you change the name in each statement inside the loop.

The `AssignIconsToSquares()` method iterates through each label control in the `TableLayoutPanel` and executes the same statements for each of them. Those statements pull a random icon from the list that you added in [Step 2: Add a Random Object and a List of Icons](#). (That's why you included two of each icon in the list, so there would be a pair of icons assigned to random label controls.)

Look more closely at the code that runs inside the `foreach` or `For Each` loop. This code is reproduced here.

```
Label iconLabel = control as Label;
if (iconLabel != null)
{
    int randomNumber = random.Next(Icons.Count);
    iconLabel.Text = Icons[randomNumber];
    // iconLabel.ForeColor = iconLabel.BackColor;
    Icons.RemoveAt(randomNumber);
}
```

```
Dim iconLabel = TryCast(control, Label)
If iconLabel IsNot Nothing Then
    Dim randomNumber = random.Next(Icons.Count)
    iconLabel.Text = Icons(randomNumber)
    ' iconLabel.ForeColor = iconLabel.BackColor
    Icons.RemoveAt(randomNumber)
End If
```

The first line converts the `control` variable to a label named `iconLabel`. The line after that is an `if` statement that checks to make sure the conversion worked. If the conversion does work, the statements in the `if` statement run. (As you may recall from the previous tutorials, the `if` statement is used to evaluate whatever condition you specify.) The first line in the `if` statement creates a variable named `randomNumber` that contains a random number that corresponds to one of the items in the `Icons` list. To do this, it uses the

`Next` method of the `Random` object that you created earlier. The `Next` method returns the random number. This line also uses the `Count` property of the `icons` list to determine the range from which to choose the random number. The next line assigns one of the icon list items to the `Text` property of the label. The commented-out line is explained later in this topic. Finally, the last line in the `if` statement removes from the list the icon that has been added to the form.

Remember, if you're not sure about what some part of the code does, you can position the mouse pointer over a code element and review the resulting tooltip. You can also step through each line of code while the program is running by using the Visual Studio debugger. See [How Do I: Step with The Debugger in Visual Studio?](#) or [Navigating through Code with the Debugger](#) for more information.

- To fill up the game board with icons, you need to call the `AssignIconsToSquares()` method as soon as the program starts. If you're using Visual C#, add a statement just below the call to the `InitializeComponent()` method in the `Form1` constructor, so your form calls your new method to set itself up before it's shown. Constructors are called when you create a new object, such as a class or struct. See [Constructors \(C# Programming Guide\)](#) or [Using Constructors and Destructors](#) in Visual Basic for more information.

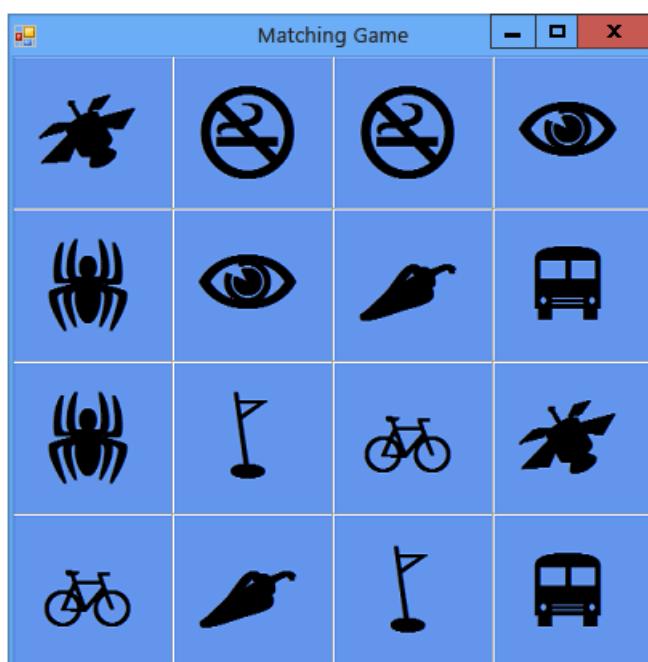
```
public Form1()
{
    InitializeComponent();

    AssignIconsToSquares();
}
```

For Visual Basic, add the `AssignIconsToSquares()` method call to the `Form1_Load` method so that the code looks like the following.

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    AssignIconsToSquares()
End Sub
```

- Save your program and run it. It should show a form with random icons assigned to each label.
- Close your program, and then run it again. Notice that different icons are assigned to each label, as shown in the following picture.



Matching game with random icons

The icons are visible now because you haven't hidden them. To hide them from the player, you can set each label's `ForeColor` property to the same color as its `BackColor` property.

TIP

Another way to hide controls like labels is to set their `Visible` property to `False`.

6. To hide the icons, stop the program and remove the comment marks for the commented line of code inside the `For Each` loop.

```
iconLabel.ForeColor = iconLabel.BackColor;
```

```
iconLabel.ForeColor = iconLabel.BackColor
```

7. On the menu bar, choose the **Save All** button to save your program, and then run it. The icons seem to have disappeared—only a blue background appears. However, the icons are randomly assigned and are still there. Because the icons are the same color as the background, it hides them from the player. After all, it wouldn't be a very challenging game if the player could see all of the icons right away!

To continue or review

- To go to the next tutorial step, see [Step 4: Add a Click Event Handler to Each Label](#).
- To return to the previous tutorial step, see [Step 2: Add a Random Object and a List of Icons](#).

Step 4: Add a Click Event Handler to Each Label

12/22/2017 • 3 min to read • [Edit Online](#)

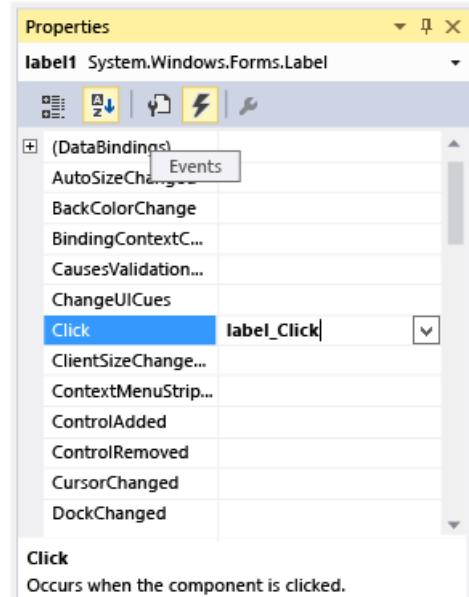
The matching game works as follows:

1. When a player chooses one of the squares with a hidden icon, the program shows the icon to the player by changing the icon color to black.
2. Then the player chooses another hidden icon.
3. If the icons match, they stay visible. If not, both icons are hidden again.

To get your program to work that way, you add a Click event handler that changes the color of the label that is chosen.

To add a Click event handler to each label

1. Open the form in the Windows Forms Designer. In Solution Explorer, choose Form1.cs or Form1.vb. On the menu bar, choose **View, Designer**.
2. Choose the first label control to select it. Then, hold down the CTRL key while you choose each of the other labels to select them. Be sure that every label is selected.
3. Choose the **Events** button on the tool bar in the **Properties** window to view the **Events** page in the **Properties** window. Scroll down to the **Click** event, and enter **label_Click** in the box, as shown in the following picture.



Properties window showing Click event

4. Choose the ENTER key. The IDE adds a Click event handler called `label_Click()` to the code, and hooks it to each of the labels on the form.
5. Fill in the rest of the code, as follows:

```

/// <summary>
/// Every label's Click event is handled by this event handler
/// </summary>
/// <param name="sender">The label that was clicked</param>
/// <param name="e"></param>
private void label_Click(object sender, EventArgs e)
{
    Label clickedLabel = sender as Label;

    if (clickedLabel != null)
    {
        // If the clicked label is black, the player clicked
        // an icon that's already been revealed --
        // ignore the click
        if (clickedLabel.ForeColor == Color.Black)
            return;

        clickedLabel.ForeColor = Color.Black;
    }
}

```

```

''' <summary>
''' Every label's Click event is handled by this event handler
''' </summary>
''' <param name="sender">The label that was clicked</param>
''' <param name="e"></param>
''' <remarks></remarks>
Private Sub label_Click(ByVal sender As System.Object,
                      ByVal e As System.EventArgs) Handles Label9.Click,
Label8.Click, Label7.Click, Label6.Click, Label5.Click, Label4.Click,
Label3.Click, Label2.Click, Label16.Click, Label15.Click, Label14.Click,
Label13.Click, Label12.Click, Label11.Click, Label10.Click, Label1.Click

    Dim clickedLabel = TryCast(sender, Label)

    If clickedLabel IsNot Nothing Then

        ' If the clicked label is black, the player clicked
        ' an icon that's already been revealed --
        ' ignore the click
        If clickedLabel.ForeColor = Color.Black Then Exit Sub

        clickedLabel.ForeColor = Color.Black
    End If
End Sub

```

NOTE

If you copy and paste the `label_Click()` code block rather than entering the code manually, be sure to replace the existing `label_Click()` code. Otherwise, you'll end up with a duplicate code block.

NOTE

You may recognize `object sender` at the top of the event handler as the same one used in the [Tutorial 2: Create a Timed Math Quiz](#) tutorial. Because you hooked up different label control Click event to a single event handler method, the same method is called no matter which label the user chooses. The event handler method needs to know which label was chosen, so it uses the name **sender** to identify the label control. The first line of the method tells the program that it's not just a generic object, but specifically a label control, and that it uses the name **clickedLabel** to access the label's properties and methods.

This method first checks whether **clickedLabel** was successfully converted (cast) from an object to a label control. If unsuccessful, it has a value of `null` (C#) or `Nothing` (Visual Basic), and you don't want to execute the remainder of the code in the method. Next, the method checks the chosen label's text color by using the label's **ForeColor** property. If the label's text color is black, then that means the icon's already been chosen and the method is done. (That's what the `return` statement does: It tells the program to stop executing the method.) Otherwise, the icon hasn't been chosen, so the program changes the label's text color to black.

6. On the menu bar, choose **File, Save All** to save your progress, and then, on the menu bar, choose **Debug, Start Debugging** to run your program. You should see an empty form with a blue background. Choose any of the cells in the form, and one of the icons should become visible. Continue choosing different places in the form. As you choose the icons, they should appear.

To continue or review

- To go to the next tutorial step, see [Step 5: Add Label References](#).
- To return to the previous tutorial step, see [Step 3: Assign a Random Icon to Each Label](#).

Step 5: Add Label References

12/22/2017 • 4 min to read • [Edit Online](#)

The program needs to track which label controls the player chooses. Right now, the program shows all labels chosen by the player. But we're going to change that. After the first label is chosen, the program should show the label's icon. After the second label is chosen, the program should display both icons for a brief time, and then hide both icons again. Your program will now keep track of which label control is chosen first and which is chosen second by using *reference variables*.

To add label references

1. Add label references to your form by using the following code.

```
Public Class Form1

    ' firstClicked points to the first Label control
    ' that the player clicks, but it will be Nothing
    ' if the player hasn't clicked a label yet
    Private firstClicked As Label = Nothing

    ' secondClicked points to the second Label control
    ' that the player clicks
    Private secondClicked As Label = Nothing
```

```
public partial class Form1 : Form
{
    // firstClicked points to the first Label control
    // that the player clicks, but it will be null
    // if the player hasn't clicked a label yet
    Label firstClicked = null;

    // secondClicked points to the second Label control
    // that the player clicks
    Label secondClicked = null;
```

These reference variables look similar to the statements you used earlier to add objects (like `Timer` objects, `List` objects, and `Random` objects) to your form. However, these statements don't cause two extra label controls to appear on the form because there's no `new` keyword used in either of the two statements. Without the `new` keyword, no object is created. That's why `firstClicked` and `secondClicked` are called reference variables: They just keep track (or, refer to) `Label` objects.

When a variable isn't keeping track of an object, it's set to a special reserved value: `null` in Visual C# and `Nothing` in Visual Basic. So, when the program starts, both `firstClicked` and `secondClicked` are set to `null` or `Nothing`, which means that the variables aren't keeping track of anything.

2. Modify your Click event handler to use the new `firstClicked` reference variable. Remove the last statement in the `label_Click()` event handler method (`clickedLabel.ForeColor = Color.Black;`) and replace it with the `if` statement that follows. (Be sure you include the comment, and the whole `if` statement.)

```

''' <summary>
''' Every label's Click event is handled by this event handler
''' </summary>
''' <param name="sender">The label that was clicked</param>
''' <param name="e"></param>
''' <remarks></remarks>
Private Sub label_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles Label9.Click,
    Label8.Click, Label7.Click, Label6.Click, Label5.Click, Label4.Click,
    Label3.Click, Label2.Click, Label16.Click, Label15.Click, Label14.Click,
    Label13.Click, Label12.Click, Label11.Click, Label10.Click, Label1.Click

    Dim clickedLabel = TryCast(sender, Label)

    If clickedLabel IsNot Nothing Then

        ' If the clicked label is black, the player clicked
        ' an icon that's already been revealed --
        ' ignore the click
        If clickedLabel.ForeColor = Color.Black Then Exit Sub

        ' If firstClicked is Nothing, this is the first icon
        ' in the pair that the player clicked,
        ' so set firstClicked to the label that the player
        ' clicked, change its color to black, and return
        If firstClicked Is Nothing Then
            firstClicked = clickedLabel
            firstClicked.ForeColor = Color.Black
            Exit Sub
        End If
    End If

End Sub

```

```

/// <summary>
/// Every label's Click event is handled by this event handler
/// </summary>
/// <param name="sender">The label that was clicked</param>
/// <param name="e"></param>
private void label_Click(object sender, EventArgs e)
{
    Label clickedLabel = sender as Label;

    if (clickedLabel != null)
    {
        // If the clicked label is black, the player clicked
        // an icon that's already been revealed --
        // ignore the click
        if (clickedLabel.ForeColor == Color.Black)
            return;

        // If firstClicked is null, this is the first icon
        // in the pair that the player clicked,
        // so set firstClicked to the label that the player
        // clicked, change its color to black, and return
        if (firstClicked == null)
        {
            firstClicked = clickedLabel;
            firstClicked.ForeColor = Color.Black;

            return;
        }
    }
}

```

3. Save and run your program. Choose one of the label controls, and its icon appears.
4. Choose the next label control, and notice that nothing happens. The program is already keeping track of the first label that the player chose, so `firstClicked` isn't equal to `null` in Visual C# or `Nothing` in Visual Basic. When your `if` statement checks `firstClicked` to determine if it's equal to `null` or `Nothing`, it finds that it isn't, and it doesn't execute the statements in the `if` statement. So, only the first icon that's chosen turns black, and the other icons are invisible, as shown in the following picture.



Matching game showing one icon

You'll fix this situation in the next step of the tutorial by adding a **Timer** control.

To continue or review

- To go to the next tutorial step, see [Step 6: Add a Timer](#).
- To return to the previous tutorial step, see [Step 4: Add a Click Event Handler to Each Label](#).

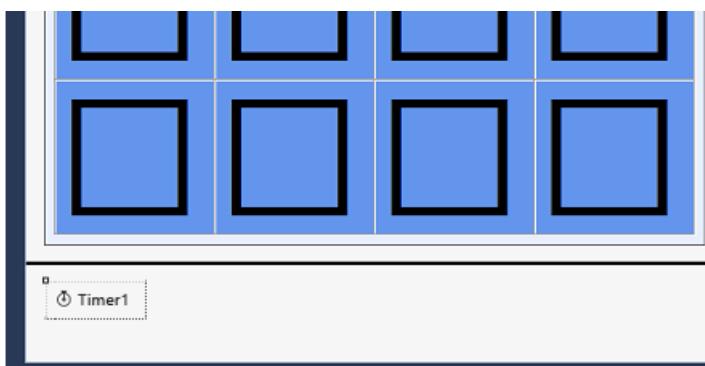
Step 6: Add a Timer

12/22/2017 • 7 min to read • [Edit Online](#)

Next, you add a **Timer** control to the matching game. A timer waits a specified number of milliseconds, and then fires an event, referred to as a *tick*. This is useful for starting an action, or repeating an action on a regular basis. In this case, you'll use a timer to enable players to choose two icons, and if the icons don't match, hide the two icons again after a short period of time.

To add a timer

1. From the toolbox in Windows Forms Designer, choose **Timer** (in the **Components** category) and then choose the ENTER key, or double-click the timer to add a timer control to the form. The timer's icon, called **Timer1**, should appear in a space below the form, as shown in the following picture.



Timer

NOTE

If the toolbox is empty, be sure to select the form designer, and not the code behind the form, before opening the toolbox.

2. Choose the **Timer1** icon to select the timer. In the **Properties** window, switch from viewing events to viewing properties. Then, set the timer's **Interval** property to **750**, but leave its **Enabled** property set to **False**. The **Interval** property tells the timer how long to wait between *ticks*, or when it triggers its **Tick** event. A value of 750 tells the timer to wait three quarters of a second (750 milliseconds) before it fires its **Tick** event. You'll call the `Start()` method to start the timer only after the player chooses the second label.
3. Choose the timer control icon in Windows Forms Designer and then choose the ENTER key, or double-click the timer, to add an empty **Tick** event handler. Either replace the code with the following code, or manually enter the following code into the event handler.

```

/// <summary>
/// This timer is started when the player clicks
/// two icons that don't match,
/// so it counts three quarters of a second
/// and then turns itself off and hides both icons
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void timer1_Tick(object sender, EventArgs e)
{
    // Stop the timer
    timer1.Stop();

    // Hide both icons
    firstClicked.ForeColor = firstClicked.BackColor;
    secondClicked.ForeColor = secondClicked.BackColor;

    // Reset firstClicked and secondClicked
    // so the next time a label is
    // clicked, the program knows it's the first click
    firstClicked = null;
    secondClicked = null;
}

```

```

''' <summary>
''' This timer is started when the player clicks
''' two icons that don't match,
''' so it counts three quarters of a second
''' and then turns itself off and hides both icons
''' </summary>
''' <remarks></remarks>
Private Sub Timer1_Tick() Handles Timer1.Tick

    ' Stop the timer
    Timer1.Stop()

    ' Hide both icons
    firstClicked.ForeColor = firstClicked.BackColor
    secondClicked.ForeColor = secondClicked.BackColor

    ' Reset firstClicked and secondClicked
    ' so the next time a label is
    ' clicked, the program knows it's the first click
    firstClicked = Nothing
    secondClicked = Nothing

End Sub

```

The Tick event handler does three things: First, it makes sure the timer isn't running by calling the `Stop()` method. Then it uses two reference variables, `firstClicked` and `secondClicked`, to make the icons of the two labels that the player chose invisible again. Finally, it resets the `firstClicked` and `secondClicked` reference variables to `null` in Visual C# and `Nothing` in Visual Basic. This step is important because it's how the program resets itself. Now it's not keeping track of any `Label` controls, and it's ready for the player to choose a label again.

NOTE

A `Timer` object has a `Start()` method that starts the timer, and a `Stop()` method that stops it. When you set the timer's **Enabled** property to **True** in the **Properties** window, it starts ticking as soon as the program begins. But when you leave it set to **False**, it doesn't start ticking until its `Start()` method is called. Normally, a timer fires its `Tick` event over and over again, using the **Interval** property to determine how many milliseconds to wait between ticks. You may have noticed how the timer's `Stop()` method is called inside the `Tick` event. That puts the timer into *one shot mode*, meaning that when the `Start()` method is called, it waits for the specified interval, triggers a single `Tick` event, and then stops.

4. To see the new timer in action, go to the code editor and add the following code to the top and bottom of the `label_Click()` event handler method. (You're adding an `if` statement to the top, and three statements to the bottom; the rest of the method stays the same.)

```
/// <summary>
/// Every label's Click event is handled by this event handler
/// </summary>
/// <param name="sender">The label that was clicked</param>
/// <param name="e"></param>
private void label_Click(object sender, EventArgs e)
{
    // The timer is only on after two non-matching
    // icons have been shown to the player,
    // so ignore any clicks if the timer is running
    if (timer1.Enabled == true)
        return;

    Label clickedLabel = sender as Label;

    if (clickedLabel != null)
    {
        // If the clicked label is black, the player clicked
        // an icon that's already been revealed --
        // ignore the click
        if (clickedLabel.ForeColor == Color.Black)
            return;

        // If firstClicked is null, this is the first icon
        // in the pair that the player clicked,
        // so set firstClicked to the label that the player
        // clicked, change its color to black, and return
        if (firstClicked == null)
        {
            firstClicked = clickedLabel;
            firstClicked.ForeColor = Color.Black;
            return;
        }

        // If the player gets this far, the timer isn't
        // running and firstClicked isn't null,
        // so this must be the second icon the player clicked
        // Set its color to black
        secondClicked = clickedLabel;
        secondClicked.ForeColor = Color.Black;

        // If the player gets this far, the player
        // clicked two different icons, so start the
        // timer (which will wait three quarters of
        // a second, and then hide the icons)
        timer1.Start();
    }
}
```

```

''' <summary>
''' Every label's Click event is handled by this event handler
''' </summary>
''' <param name="sender">The label that was clicked</param>
''' <param name="e"></param>
''' <remarks></remarks>
Private Sub label_Click(ByVal sender As System.Object,
                      ByVal e As System.EventArgs) Handles Label9.Click,
Label8.Click, Label7.Click, Label6.Click, Label5.Click, Label4.Click,
Label3.Click, Label2.Click, Label16.Click, Label15.Click, Label14.Click,
Label13.Click, Label12.Click, Label11.Click, Label10.Click, Label1.Click

    ' The timer is only on after two non-matching
    ' icons have been shown to the player,
    ' so ignore any clicks if the timer is running
    If Timer1.Enabled Then Exit Sub

    Dim clickedLabel = TryCast(sender, Label)

    If clickedLabel IsNot Nothing Then
        ' If the clicked label is black, the player clicked
        ' an icon that's already been revealed --
        ' ignore the click
        If clickedLabel.ForeColor = Color.Black Then Exit Sub

        ' If firstClicked is Nothing, this is the first icon
        ' in the pair that the player clicked,
        ' so set firstClicked to the label that the player
        ' clicked, change its color to black, and return
        If firstClicked Is Nothing Then
            firstClicked = clickedLabel
            firstClicked.ForeColor = Color.Black
            Exit Sub
        End If

        ' If the player gets this far, the timer isn't
        ' running and firstClicked isn't Nothing,
        ' so this must be the second icon the player clicked
        ' Set its color to black
        secondClicked = clickedLabel
        secondClicked.ForeColor = Color.Black

        ' If the player gets this far, the player
        ' clicked two different icons, so start the
        ' timer (which will wait three quarters of
        ' a second, and then hide the icons)
        Timer1.Start()
    End If

End Sub

```

The code at the top of the method checks whether the timer was started by checking the value of the **Enabled** property. That way, if the player chooses the first and second `Label` controls and the timer starts, choosing a third label won't do anything.

The code at the bottom of the method sets the `secondClicked` reference variable to track the second `Label` control that the player chose, and then it sets that label's icon color to black to make it visible. Then, it starts the timer in one shot mode, so that it waits 750 milliseconds and then fires a single Tick event. The timer's Tick event handler hides the two icons and resets the `firstClicked` and `secondClicked` reference variables so the form is ready for the player to choose another pair of icons.

5. Save and run your program. Choose an icon, and it becomes visible.
6. Choose another icon. It appears briefly, and then both icons disappear. Repeat this numerous times. The

form now keeps track of the first and second icons that you choose, and uses the timer to pause before making the icons disappear.

To continue or review

- To go to the next tutorial step, see [Step 7: Keep Pairs Visible](#).
- To return to the previous tutorial step, see [Step 5: Add Label References](#).

Step 7: Keep Pairs Visible

12/22/2017 • 3 min to read • [Edit Online](#)

The game works well, as long as the player only chooses pairs of icons that don't match. But consider what should happen when the player chooses a matching pair. Instead of making the icons disappear by turning on the timer (using the `Start()` method), the game should reset itself so that it's no longer keeping track of any labels using the `firstClicked` and `secondClicked` reference variables, without resetting the colors for the two labels that were chosen.

To keep pairs visible

1. Add the following `if` statement to the `label_Click()` event handler method, near the end of the code just above the statement where you start the timer. Take a close look at the code while adding it to the program. Consider how the code works.

```
// If the player gets this far, the timer isn't
// running and firstClicked isn't null,
// so this must be the second icon the player clicked
// Set its color to black
secondClicked = clickedLabel;
secondClicked.ForeColor = Color.Black;

// If the player clicked two matching icons, keep them
// black and reset firstClicked and secondClicked
// so the player can click another icon
if (firstClicked.Text == secondClicked.Text)
{
    firstClicked = null;
    secondClicked = null;
    return;
}

// If the player gets this far, the player
// clicked two different icons, so start the
// timer (which will wait three quarters of
// a second, and then hide the icons)
timer1.Start();
}
```

```

' If the player gets this far, the timer isn't
' running and firstClicked isn't Nothing,
' so this must be the second icon the player clicked
' Set its color to black
secondClicked = clickedLabel
secondClicked.ForeColor = Color.Black

' If the player clicked two matching icons, keep them
' black and reset firstClicked and secondClicked
' so the player can click another icon
If firstClicked.Text = secondClicked.Text Then
    firstClicked = Nothing
    secondClicked = Nothing
    Exit Sub
End If

' If the player gets this far, the player
' clicked two different icons, so start the
' timer (which will wait three quarters of
' a second, and then hide the icons)
Timer1.Start()
End If
End Sub

```

The first line of the `if` statement you just added checks whether the icon in the first label that the player chooses is the same as the icon in the second label. If the icons are identical, the program executes the three statements between the curly braces in C# or the three statements within the `if` statement in Visual Basic. The first two statements reset the `firstClicked` and `secondClicked` reference variables so that they no longer keep track of any of the labels. (You may recognize those two statements from the timer's Tick event handler.) The third statement is a `return` statement, which tells the program to skip the rest of the statements in the method without executing them.

If programming in Visual C#, you may have noticed that some of the code uses a single equal sign (`=`), while other statements use two equal signs (`==`). Consider why `=` is used in some places but `==` is used in other places.

This is a good example that shows the difference. Take a careful look at the code between the parentheses in the `if` statement.

```
firstClicked.Text = secondClicked.Text
```

```
firstClicked.Text == secondClicked.Text
```

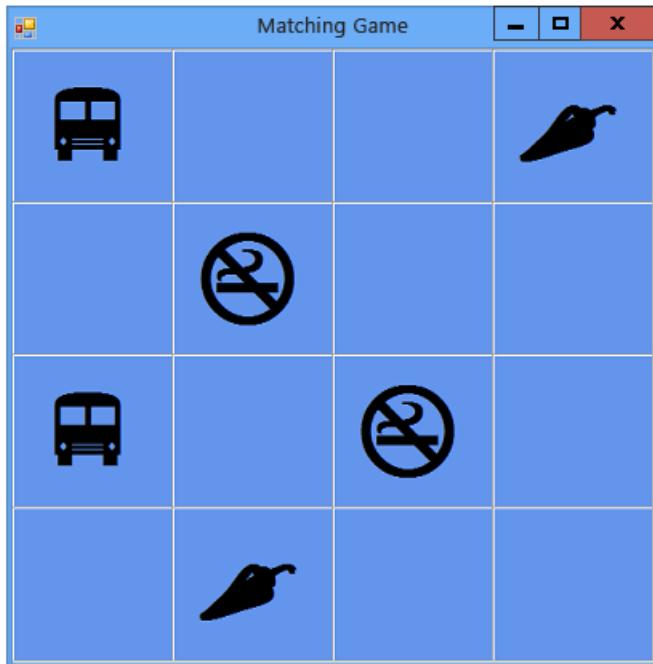
Then look closely at the first statement in the block of code after the `if` statement.

```
firstClicked = Nothing
```

```
firstClicked = null;
```

The first of those two statements checks whether two icons are the same. Because two values are being compared, the Visual C# program uses the `==` equality operator. The second statement actually changes the value (called *assignment*), setting the `firstClicked` reference variable equal to `null` to reset it. That's why it uses the `=` assignment operator instead. Visual C# uses `=` to set values, and `==` to compare them. Visual Basic uses `=` for both variable assignment and comparison.

2. Save and run the program, and then start choosing icons on the form. If you choose a pair that doesn't match, the timer's Tick event triggers, and both icons disappear. If you choose a matching pair, the new `if` statement executes, and the return statement causes the method to skip the code that starts the timer, so the icons stay visible, as shown in the following picture.



Matching game with visible icon pairs

To continue or review

- To go to the next tutorial step, see [Step 8: Add a Method to Verify Whether the Player Won](#).
- To return to the previous tutorial step, see [Step 6: Add a Timer](#).

Step 8: Add a Method to Verify Whether the Player Won

12/22/2017 • 3 min to read • [Edit Online](#)

You've created a fun game, but it needs an additional item to finish it. The game should end when the player wins, so you need to add a `CheckForWinner()` method to verify whether the player won.

To add a method to verify whether the player won

1. Add a `CheckForWinner()` method to the bottom of your code, below the `timer1_Tick()` event handler, as shown in the following code.

```
/// <summary>
/// Check every icon to see if it is matched, by
/// comparing its foreground color to its background color.
/// If all of the icons are matched, the player wins
/// </summary>
private void CheckForWinner()
{
    // Go through all of the labels in the TableLayoutPanel,
    // checking each one to see if its icon is matched
    foreach (Control control in tableLayoutPanel1.Controls)
    {
        Label iconLabel = control as Label;

        if (iconLabel != null)
        {
            if (iconLabel.ForeColor == iconLabel.BackColor)
                return;
        }
    }

    // If the loop didn't return, it didn't find
    // any unmatched icons
    // That means the user won. Show a message and close the form
    MessageBox.Show("You matched all the icons!", "Congratulations");
    Close();
}
```

```

''' <summary>
''' Check every icon to see if it is matched, by
''' comparing its foreground color to its background color.
''' If all of the icons are matched, the player wins
''' </summary>
Private Sub CheckForWinner()

    ' Go through all of the labels in the TableLayoutPanel,
    ' checking each one to see if its icon is matched
    For Each control In TableLayoutPanel1.Controls
        Dim iconLabel = TryCast(control, Label)
        If iconLabel IsNot Nothing AndAlso
            iconLabel.ForeColor = iconLabel.BackColor Then Exit Sub
    Next

    ' If the loop didn't return, it didn't find
    ' any unmatched icons
    ' That means the user won. Show a message and close the form
    MessageBox.Show("You matched all the icons!", "Congratulations")
    Close()

End Sub

```

The method uses another `foreach` loop in Visual C# or `For Each` loop in Visual Basic to go through each label in the `TableLayoutPanel`. It uses the equality operator (`==` in Visual C# and `=` in Visual Basic) to check each label's icon color to verify whether it matches the background. If the colors match, the icon remains invisible, and the player hasn't matched all of the icons remaining. In that case, the program uses a `return` statement to skip the rest of the method. If the loop gets through all of the labels without executing the `return` statement, that means that all of the icons on the form were matched. The program shows a `MessageBox` to congratulate the player on winning, and then calls the form's `Close()` method to end the game.

2. Next, have the label's Click event handler call the new `CheckForWinner()` method. Be sure that your program checks for a winner immediately after it shows the second icon that the player chooses. Look for the line where you set the second chosen icon's color, and then call the `CheckForWinner()` method right after that, as shown in the following code.

```

// If the player gets this far, the timer isn't
// running and firstClicked isn't null,
// so this must be the second icon the player clicked
// Set its color to black
secondClicked = clickedLabel;
secondClicked.ForeColor = Color.Black;

// Check to see if the player won
CheckForWinner();

// If the player clicked two matching icons, keep them
// black and reset firstClicked and secondClicked
// so the player can click another icon
if (firstClicked.Text == secondClicked.Text)
{
    firstClicked = null;
    secondClicked = null;
    return;
}

```

```

' If the player gets this far, the timer isn't
' running and firstClicked isn't Nothing,
' so this must be the second icon the player clicked
' Set its color to black
secondClicked = clickedLabel
secondClicked.ForeColor = Color.Black

' Check to see if the player won
CheckForWinner()

' If the player clicked two matching icons, keep them
' black and reset firstClicked and secondClicked
' so the player can click another icon
If firstClicked.Text = secondClicked.Text Then
    firstClicked = Nothing
    secondClicked = Nothing
    Exit Sub
End If

```

3. Save and run the program. Play the game and match all of the icons. When you win, the program displays a congratulatory MessageBox (as shown in the following picture), and then closes the box.



Matching game with MessageBox

To continue or review

- To go to the next tutorial step, see [Step 9: Try Other Features](#).
- To return to the previous tutorial step, see [Step 7: Keep Pairs Visible](#).

Step 9: Try Other Features

1/26/2018 • 1 min to read • [Edit Online](#)

To learn more, try changing icons and colors, adding a game timer, and adding sounds. To make the game more challenging, try making the board bigger and adjusting the timer.

To download a completed version of the sample, see [Complete Matching Game tutorial sample](#).

To try other features

- Replace the icons and colors with ones you choose.

TIP

Try looking at the label's `Forecolor` property.

- Add a game timer that tracks how long it takes for the player to win.

TIP

To do this, you can add a label to display the elapsed time on the form above the `TableLayoutPanel`, and add another timer to the form to track the time. Use code to start the timer when the player starts the game, and stop the timer after they match the last two icons.

- Add a sound when the player finds a match, another sound when the player uncovers two icons that don't match, and a third sound when the program hides the icons again.

TIP

To play sounds, you can use the `System.media` namespace. See [Play Sounds in Windows Forms App \(C#\)](#) or [How To Play Audio In Visual Basic](#) for more information.

- Make the game more difficult by making the board bigger.

TIP

You'll need to do more than just add rows and columns to the `TableLayoutPanel` - you'll also need to consider the number of icons you create.

- Make the game more challenging by hiding the first icon if the player is too slow to respond and doesn't choose the second icon before a certain amount of time.

To continue or review

- If you get stuck or have programming questions, try posting your question on one of the MSDN forums. See [Visual Basic Forum](#) and [Visual C# Forum](#).
- There are great, free video learning resources available to you. To learn more about programming in Visual Basic, see [Visual Basic Fundamentals: Development for Absolute Beginners](#). To learn more about programming in Visual C#, see [C# Fundamentals: Development for Absolute Beginners](#).
- To return to the previous tutorial step, see [Step 8: Add a Method to Verify Whether the Player Won](#).

Accessing data in Visual Studio

3/22/2018 • 7 min to read • [Edit Online](#)

In Visual Studio, you can create applications that connect to data in virtually any database product or service, in any format, anywhere—on a local machine, on a local area network, or in a public, private, or hybrid cloud.

For applications in JavaScript, Python, PHP, Ruby, or C++, you connect to data like you do anything else, by obtaining libraries and writing code. For .NET applications, Visual Studio provides tools that you can use to explore data sources, create object models to store and manipulate data in memory, and bind data to the user interface. Microsoft Azure provides SDKs for .NET, Java, Node.js, PHP, Python, Ruby, and mobile apps, and tools in Visual Studio for connecting to Azure Storage.

The following lists show just a few of the many database and storage systems that can be used from Visual Studio. The [Microsoft Azure](#) offerings are data services that include all provisioning and administration of the underlying data store. [Azure Tools for Visual Studio](#) is an optional component that enables you to work with Azure data stores directly from Visual Studio. Most of the other SQL and NoSQL database products that are listed here can be hosted on a local machine, on a local network, or in Microsoft Azure on a virtual machine. In this scenario, you are responsible for managing the database itself.

Microsoft Azure

| | | |
|--------------------|-----------------------------|--|
| SQL Database | Azure Cosmos DB | Storage (blobs, tables, queues, files) |
| SQL Data Warehouse | SQL Server Stretch Database | StorSimple |

And more...

SQL

| | | |
|---|----------|------------|
| SQL Server 2005-2016, including Express and LocalDB | Firebird | MariaDB |
| MySQL | Oracle | PostgreSQL |
| SQLite | | |

And more...

NoSQL

| | | |
|------------------|----------|---------|
| Apache Cassandra | CouchDB | MongoDB |
| NDatabase | OrientDB | RavenDB |
| VelocityDB | | |

And more...

Many database vendors and third parties support Visual Studio integration by NuGet packages. You can explore the offerings on nuget.org or through the NuGet Package Manager in Visual Studio (**Tools > NuGet Package Manager > Manage NuGet Packages for Solution**). Other database products integrate with Visual Studio as an extension. You can browse these offerings in the Visual Studio Marketplace by navigating to **Tools, Extensions and Updates** and then selecting **Online** in the left pane of the dialog box. For more information, see [Compatible database systems for Visual Studio](#).

NOTE

Extended support for SQL Server 2005 ended on April 12, 2016. There is no guarantee that data tools in Visual Studio 2015 and later will continue to work with SQL Server 2005 after this date. For more information, see the [end-of-support announcement for SQL Server 2005](#).

.NET languages

All .NET data access, including in .NET Core, is based on ADO.NET, a set of classes that defines an interface for accessing any kind of data source, both relational and non-relational. Visual Studio has several tools and designers that work with ADO.NET to help you connect to databases, manipulate the data, and present the data to the user. The documentation in this section describes how to use those tools. You can also program directly against the ADO.NET command objects. For more information about calling the ADO.NET APIs directly, see [ADO.NET](#).

For data-access documentation specifically related to ASP.NET, see [Working with Data](#) on the ASP.NET site. For a tutorial on using Entity Framework with ASP.NET MVC, see [Getting Started with Entity Framework 6 Code First using MVC 5](#).

Universal Windows Platform (UWP) apps in C# or Visual Basic can use the Microsoft Azure SDK for .NET to access Azure Storage and other Azure services. The Windows.Web.HttpClient class enables communication with any RESTful service. For more information, see [How to connect to an HTTP server using Windows.Web.Http](#).

For data storage on the local machine, the recommended approach is to use SQLite, which runs in the same process as the app. If an object-relational mapping (ORM) layer is required, you can use Entity Framework. For more information, see [Data access](#) in the Windows Developer Center.

If you are connecting to Azure services, be sure to download the latest [Azure SDK tools](#).

Data providers

For a database to be consumable in ADO.NET, it must have a custom *ADO.NET data provider* or else must expose an ODBC or OLE DB interface. Microsoft provides a [list of ADO.NET data providers](#) for SQL Server products, as well as ODBC and OLE DB providers.

Data modeling

In .NET, you have three choices for modeling and manipulating data in memory after you have retrieved it from a data source:

Entity Framework The preferred Microsoft ORM technology. You can use it to program against relational data as first-class .NET objects. For new applications, it should be the default first choice when a model is required. It requires custom support from the underlying ADO.NET provider.

LINQ to SQL An earlier-generation object-relational mapper. It works well for less complex scenarios but is no longer in active development.

Datasets The oldest of the three modeling technologies. It is designed primarily for rapid development of "forms over data" applications in which you are not processing huge amounts of data or performing complex queries or transformations. A DataSet object consists of DataTable and DataRow objects that logically resemble SQL database objects much more than .NET objects. For relatively simple applications based on SQL data sources, datasets might

still be a good choice.

There is no requirement to use any of these technologies. In some scenarios, especially where performance is critical, you can simply use a DataReader object to read from the database and copy the values that you need into a collection object such as List<T>.

Native C++

C++ applications that connect to SQL Server should use the [Microsoft® ODBC Driver 13.1 for SQL Server](#) in most cases. If the servers are linked, then OLE DB is necessary and for that you use the [SQL Server Native Client](#). You can access other databases by using [ODBC](#) or OLE DB drivers directly. ODBC is the current standard database interface, but most database systems provide custom functionality that can't be accessed through the ODBC interface. OLE DB is a legacy COM data-access technology that is still supported but not recommended for new applications. For more information, see [Data Access in Visual C++](#).

C++ programs that consume REST services can use the [C++ REST SDK](#).

C++ programs that work with Microsoft Azure Storage can use the [Microsoft Azure Storage Client](#).

Data modeling—Visual Studio does not provide an ORM layer for C++. [ODB](#) is a popular open-source ORM for C++.

To learn more about connecting to databases from C++ apps, see [Visual Studio data tools for C++](#). For more information about legacy Visual C++ data-access technologies, see [Data Access](#).

JavaScript

[JavaScript in Visual Studio](#) is a first-class language for building cross-platform apps, UWP apps, cloud services, websites, and web apps. You can use Bower, Grunt, Gulp, npm, and NuGet from within Visual Studio to install your favorite JavaScript libraries and database products. Connect to Azure storage and services by downloading SDKs from the [Azure website](#). Edge.js is a library that connects server-side JavaScript (Node.js) to ADO.NET data sources.

Python

Install [Python support in Visual Studio](#) to create Python applications. The Azure documentation has several tutorials on connecting to data, including the following:

- [Django and SQL Database on Azure](#)
- [Django and MySQL on Azure](#)
- Work with [blobs](#), [files](#), [queues](#), and [tables \(Cosmo DB\)](#).

Related topics

[Microsoft AI platform](#)—Provides an introduction to the Microsoft intelligent cloud, including Cortana Analytics Suite and support for Internet of Things.

[Microsoft Azure Storage](#)—Describes Azure Storage, and how to create applications by using Azure blobs, tables, queues, and files.

[Azure SQL Database](#)—Describes how to connect to Azure SQL Database, a relational database as a service.

[SQL Server Data Tools](#)—Describes the tools that simplify design, exploration, testing, and deploying of data-connected applications and databases.

[ADO.NET](#)—Describes the ADO.NET architecture and how to use the ADO.NET classes to manage application data and interact with data sources and XML.

[ADO.NET Entity Framework](#)—Describes how to create data applications that allow developers to program against a conceptual model instead of directly against a relational database.

[WCF Data Services 4.5](#)—Describes how to use WCF Data Services to deploy data services on the web or an intranet that implement the [Open Data Protocol \(OData\)](#).

[Data in Office Solutions](#)—Contains links to topics that explain how data works in Office solutions. This includes information about schema-oriented programming, data caching, and server-side data access.

[LINQ \(Language-Integrated Query\)](#)—Describes the query capabilities built into C# and Visual Basic, and the common model for querying relational databases, XML documents, datasets, and in-memory collections.

[XML Tools in Visual Studio](#)—Discusses working with XML data, debugging XSLT, .NET Framework XML features, and the architecture of XML Query.

[XML Documents and Data](#)—Provides an overview to a comprehensive and integrated set of classes that work with XML documents and data in the .NET Framework.

Designing User Interfaces

12/22/2017 • 1 min to read • [Edit Online](#)

You can create and design the user interface for your application by using a variety of tools in Visual Studio.

| TO LEARN MORE ABOUT | SEE |
|---|---|
| The features of the XAML designers in Visual Studio and Blend for Visual Studio | Designing XAML in Visual Studio and Blend for Visual Studio |
| Designing any XAML-based app using Visual Studio | Creating a UI by using XAML Designer in Visual Studio |
| Designing any XAML-based app using Blend for Visual Studio | Creating a UI by using Blend for Visual Studio |
| Designing desktop applications that use the WPF flavor of XAML | Create Modern Desktop Applications with Windows Presentation Foundation |
| Developing a DirectX application in Visual Studio | Working with 3-D Assets for Games and Apps |
| Standard icons available for your programs | The Visual Studio Image Library |

Compiling and building in Visual Studio

12/22/2017 • 2 min to read • [Edit Online](#)

Running a build creates assemblies and executable applications from your source code at any point during a development cycle. In general, the build process is very similar across many different project types such as Windows, ASP.NET, mobile apps, and others. The build process is also very similar across programming languages such as C#, Visual Basic, C++, and F#.

By building your code often, you can quickly identify compile-time errors, such as incorrect syntax, misspelled keywords, and type mismatches. You can also quickly detect and correct run-time errors, such as logic errors and semantic errors, by frequently building and running debug versions of the code.

A successful build is essentially a validation that the application's source code contains correct syntax and that all static references to libraries, assemblies, and other components have been resolved. This produces an application executable that can then be tested for proper functioning in both a [debugging environment](#) and through a variety of manual and automated tests to [validate code quality](#). Once the application has been fully tested, you can then compile a release version to deploy to your customers. For an introduction to this process, see [Walkthrough: Building an Application](#).

Within the Visual Studio product family, there are three methods you can use to build an application: the Visual Studio IDE, the MSBuild command-line tools, and Team Foundation Build on Visual Studio Team Services:

| BUILD METHOD | BENEFITS |
|-----------------------|---|
| IDE | <ul style="list-style-type: none">- Create builds immediately and test them in a debugger.- Run multi-processor builds for C++ and C# projects.- Customize different aspects of the build system. |
| MSBuild command line | <ul style="list-style-type: none">- Build projects without installing Visual Studio.- Run multi-processor builds for all project types.- Customize most areas of the build system. |
| Team Foundation Build | <ul style="list-style-type: none">- Automate your build process as part of a continuous integration/continuous delivery pipeline.- Apply automated tests with every build.- Employ virtually unlimited cloud-based resources for build processes.- Modify the build workflow and create build activities to perform deeply customized tasks. |

The documentation in this section goes into further details of the IDE-based build process. For more information on the other methods, see [MSBuild](#) and [Continuous integration and deployment](#), respectively.

Overview of building from the IDE

When you create a project, Visual Studio creates default build configurations for the project and the solution that contains the project. These configurations define how the solutions and projects are built and deployed. Project configurations in particular are unique for a target platform (such as Windows or Linux) and build type (such as debug or release). You can edit these configurations however you like, and can also create your own configurations as needed.

For a first introduction to building within the IDE, see [Walkthrough: Building an Application](#).

Next, see [Building and cleaning projects and solutions in Visual Studio](#) to learn about the different aspects customizations you can make to the process. Customizations include [changing output directories](#), [specifying custom build events](#), [managing project dependencies](#), [managing build log files](#), and [suppressing compiler warnings](#).

From there, you can explore a variety of other tasks:

- [Understand build configurations](#)
- [Understand build platforms](#)
- [Manage Project and Solution Properties.](#)
- [Specify build events in C# and Visual Basic.](#)
- [Set build options](#)
- [Build Multiple Projects in Parallel.](#)

See Also

- [Building \(Compiling\) Web Site Projects](#)

Walkthrough: Building an Application

1/30/2018 • 5 min to read • [Edit Online](#)

By completing this walkthrough, you'll become more familiar with several options that you can configure when you build applications with Visual Studio. You'll create a custom build configuration, hide certain warning messages, and increase build output information for a sample application.

Install the Sample Application

Download the [Introduction to Building WPF Applications](#) sample. Choose either C# or Visual Basic. After the .zip file has downloaded, extract it and open the **ExpenselIntro.sln** file using Visual Studio.

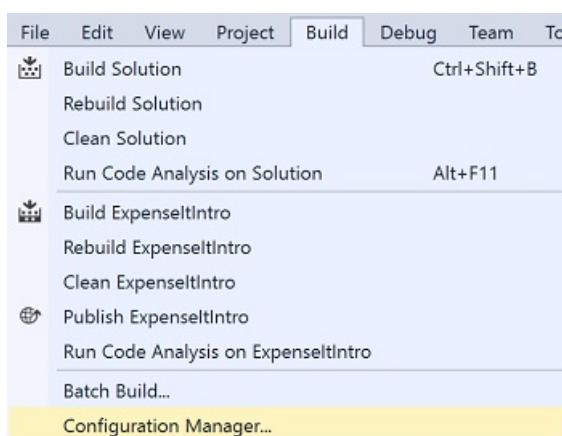
Create a Custom Build Configuration

When you create a solution, debug and release build configurations and their default platform targets are defined for the solution automatically. You can then customize these configurations or create your own. Build configurations specify the build type. Build platforms specify the operating system that an application targets for that configuration. For more information, see [Understanding Build Configurations](#), [Understanding Build Platforms](#), and [How to: Set debug and release configurations](#).

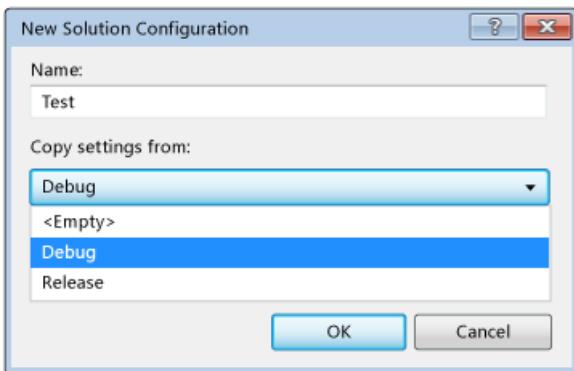
You can change or create configurations and platform settings by using the **Configuration Manager** dialog box. In this procedure, you'll create a build configuration for testing.

To create a build configuration

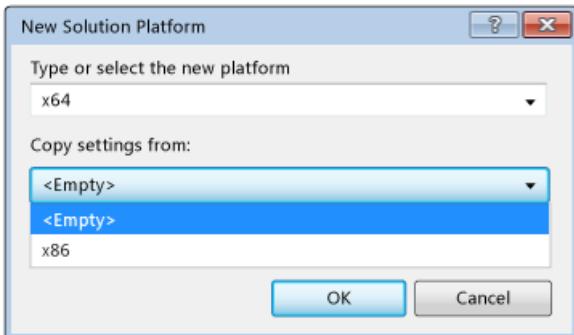
1. Open the **Configuration Manager** dialog box.



2. In the **Active solution configuration** list, choose <New...>.
3. In the **New Solution Configuration** dialog box, name the new configuration **Test**, copy settings from the existing Debug configuration, and then choose the **OK** button.

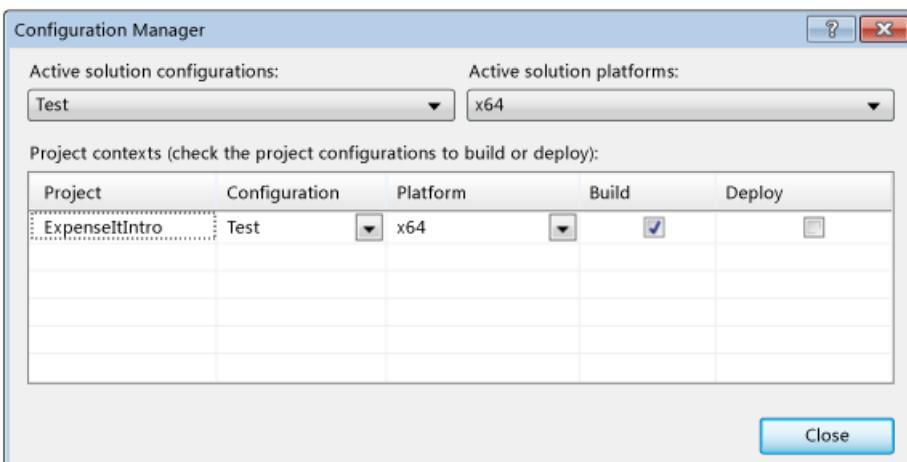


4. In the **Active solution platform** list, choose <New...>.
5. In the **New Solution Platform** dialog box, choose **x64**, and don't copy settings from the x86 platform.



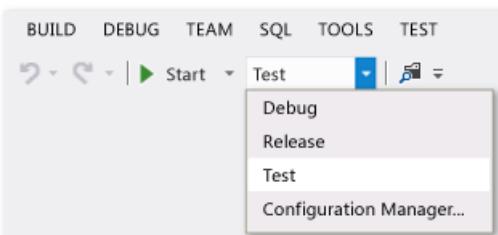
6. Choose the **OK** button.

The active solution configuration has been changed to Test with the active solution platform set to x64.



7. Choose **Close**.

You can quickly verify or change the active solution configuration by using the **Solution Configurations** list on the **Standard** toolbar.



Build the Application

Next, you'll build the solution with the custom build configuration.

To build the solution

- On the menu bar, choose **Build > Build Solution**.

The **Output** window displays the results of the build. The build succeeded.

Hide Compiler Warnings

Next we'll introduce some code that causes a warning to be generated by the compiler.

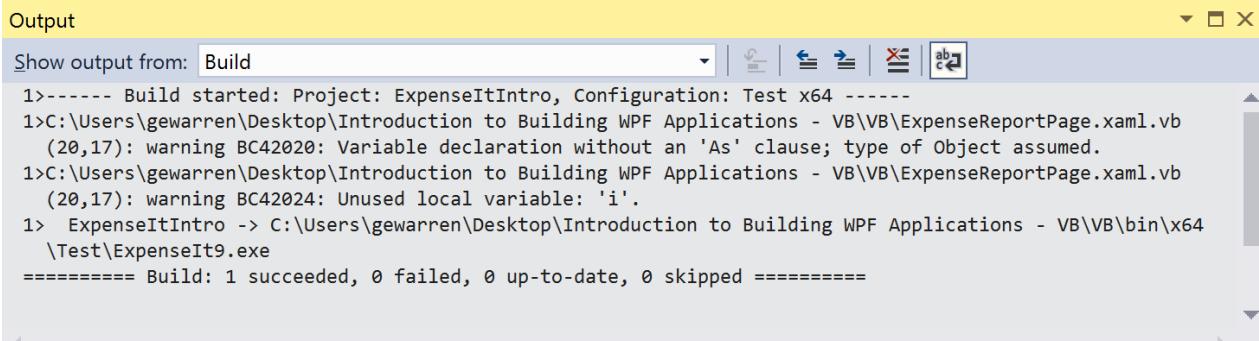
- In the C# project, open the **ExpenseReportPage.xaml.cs** file. In the **ExpenseReportPage** method, add the following code: `int i;`.

OR

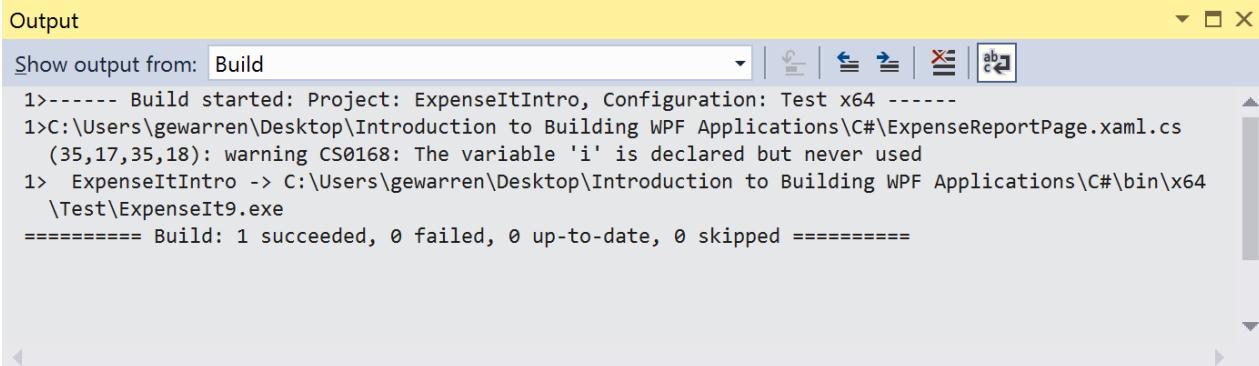
In the Visual Basic project, open the **ExpenseReportPage.xaml.vb** file. In the custom constructor **Public Sub New...**, add the following code: `Dim i`.

- Build the solution.

The **Output** window displays the results of the build. The build succeeded, but warnings were generated:



```
Output
Show output from: Build
1>----- Build started: Project: ExpenseItIntro, Configuration: Test x64 -----
1>C:\Users\gewarren\Desktop\Introduction to Building WPF Applications - VB\VB\ExpenseReportPage.xaml.vb
(20,17): warning BC42020: Variable declaration without an 'As' clause; type of Object assumed.
1>C:\Users\gewarren\Desktop\Introduction to Building WPF Applications - VB\VB\ExpenseReportPage.xaml.vb
(20,17): warning BC42024: Unused local variable: 'i'.
1> ExpenseItIntro -> C:\Users\gewarren\Desktop\Introduction to Building WPF Applications - VB\VB\bin\x64
\Test\ExpenseIt9.exe
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```



```
Output
Show output from: Build
1>----- Build started: Project: ExpenseItIntro, Configuration: Test x64 -----
1>C:\Users\gewarren\Desktop\Introduction to Building WPF Applications\C#\ExpenseReportPage.xaml.cs
(35,17,35,18): warning CS0168: The variable 'i' is declared but never used
1> ExpenseItIntro -> C:\Users\gewarren\Desktop\Introduction to Building WPF Applications\C#\bin\x64
\Test\ExpenseIt9.exe
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

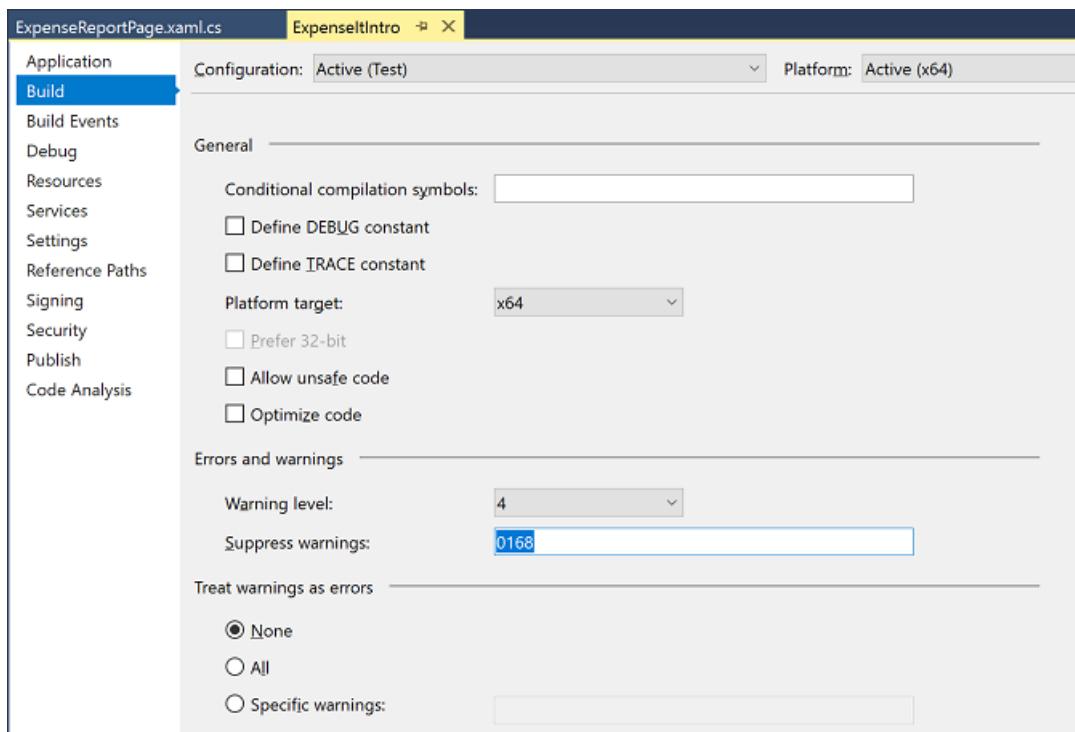
You can temporarily hide certain warning messages during a build rather than have them clutter up the build output.

To hide a specific C# warning

- In **Solution Explorer**, choose the top-level project node.
- On the menu bar, choose **View, Property Pages**.

The **Project Designer** opens.

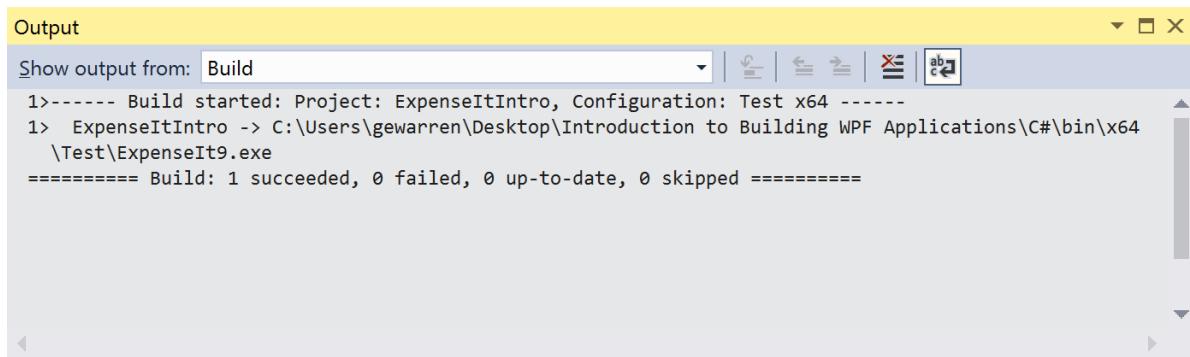
- Choose the **Build** page and then, in the **Suppress warnings** box, specify the warning number **0168**.



For more information, see [Build Page, Project Designer \(C#\)](#).

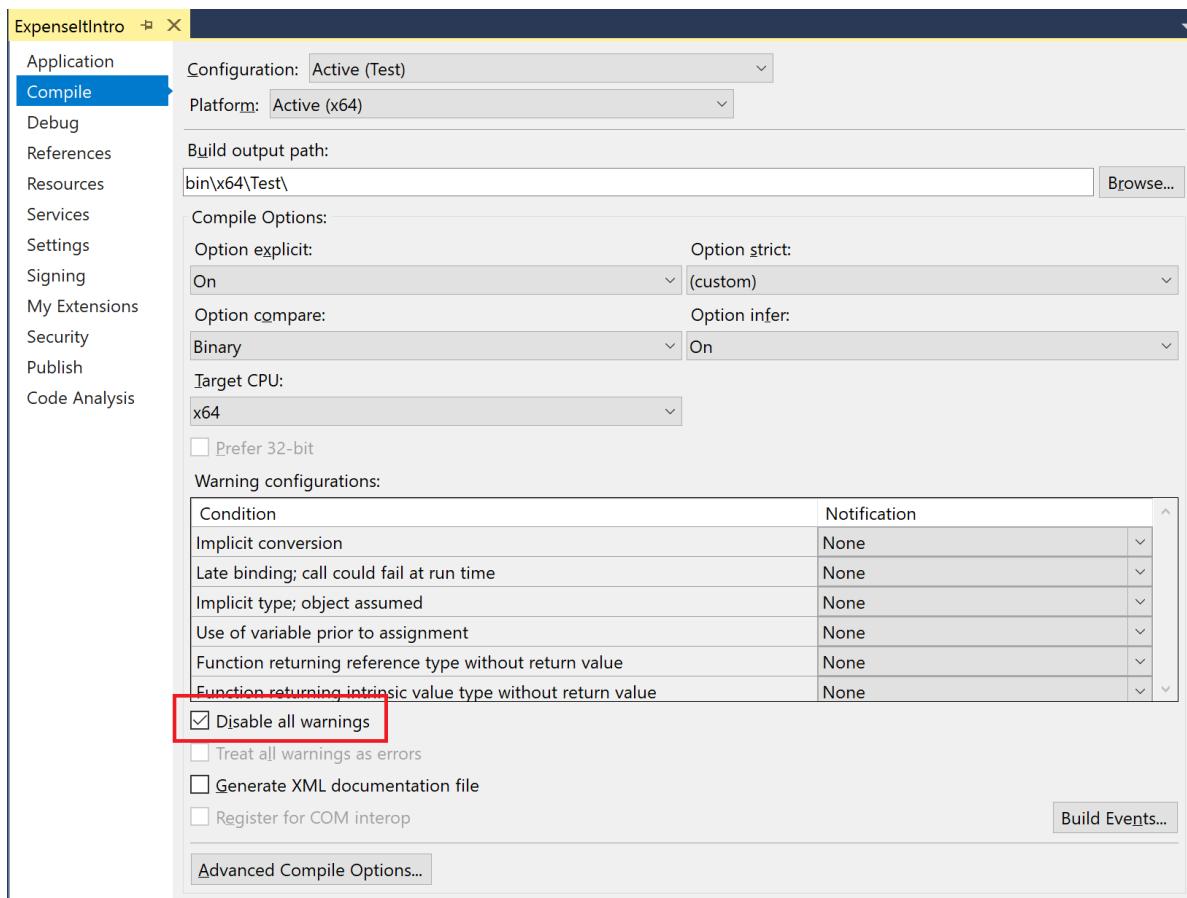
4. Build the solution.

The **Output** window displays only summary information for the build.



To suppress all Visual Basic build warnings

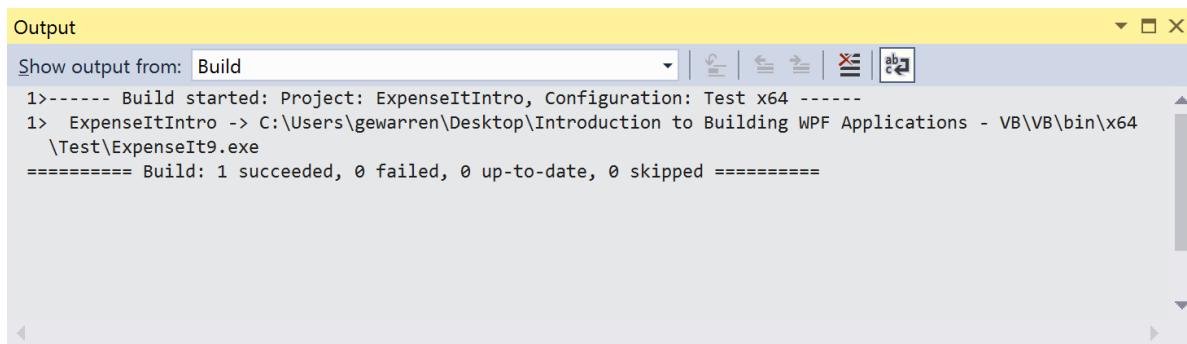
1. In **Solution Explorer**, choose the top-level project node.
2. On the menu bar, choose **View, Property Pages**.
The **Project Designer** opens.
3. On the **Compile** page, select the **Disable all warnings** check box.



For more information, see [Configuring Warnings in Visual Basic](#).

4. Build the solution.

The **Output** window displays only summary information for the build.



For more information, see [How to: Suppress Compiler Warnings](#).

Display Additional Build Details in the Output Window

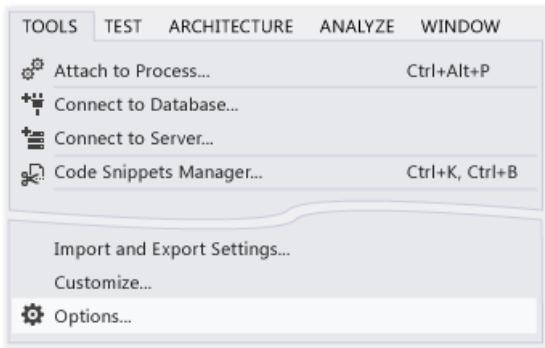
You can change how much information about the build process appears in the **Output** window. Build verbosity is usually set to Minimal, which means that the **Output** window displays only a summary of the build process along with any high priority warnings or errors. You can display more information about the build by using the [Options Dialog Box, Projects and Solutions, Build and Run](#).

IMPORTANT

If you display more information, the build will take longer to complete.

To change the amount of information in the Output window

1. Open the **Options** dialog box.



2. Choose the **Projects and Solutions** category, and then choose the **Build and Run** page.
3. In the **MSBuild project build output verbosity** list, choose **Normal**, and then choose the **OK** button.
4. On the menu bar, choose **Build**, **Clean Solution**.
5. Build the solution, and then review the information in the **Output** window.

The build information includes the time that the build started (located at the beginning) and the order in which files were processed. This information also includes the actual compiler syntax that Visual Studio runs during the build.

For example, in the C# build, the `/nowarn` option lists the warning code, 1762, that you specified earlier in this topic, along with three other warnings.

In the Visual Basic build, `/nowarn` doesn't include specific warnings to exclude, so no warnings appear.

TIP

You can search the contents of the **Output** window if you display the **Find** dialog box by choosing the **Ctrl+F** keys.

For more information, see [How to: View, Save, and Configure Build Log Files](#).

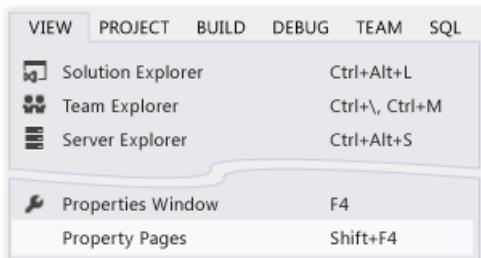
Create a Release Build

You can build a version of the sample application that's optimized for shipping it. For the release build, you'll specify that the executable is copied to a network share before the build is kicked off.

For more information, see [How to: Change the Build Output Directory](#) and [Building and Cleaning Projects and Solutions in Visual Studio](#).

To specify a release build for Visual Basic

1. Open the **Project Designer**.



2. Choose the **Compile** page.
3. In the **Configuration** list, choose **Release**.
4. In the **Platform** list, choose **x86**.

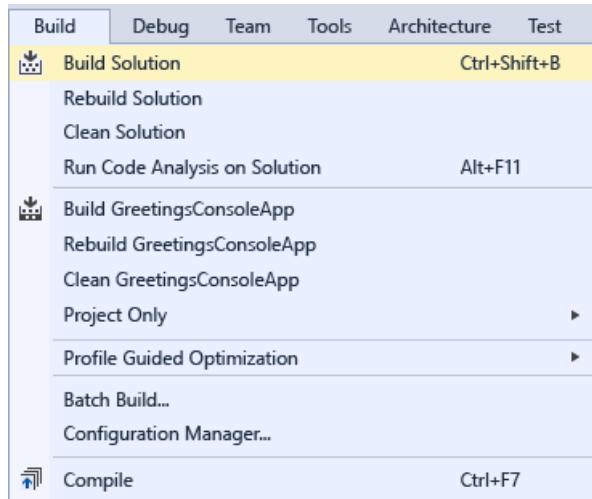
5. In the **Build output path** box, specify a network path.

For example, you can specify \\myserver\builds.

IMPORTANT

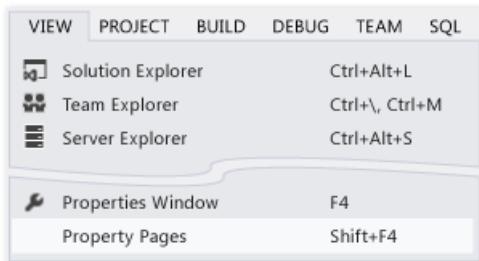
A message box might appear, warning you that the network share that you've specified might not be a trusted location. If you trust the location that you've specified, choose the **OK** button in the message box.

6. Build the application.



To specify a release build for C#

1. Open the **Project Designer**.



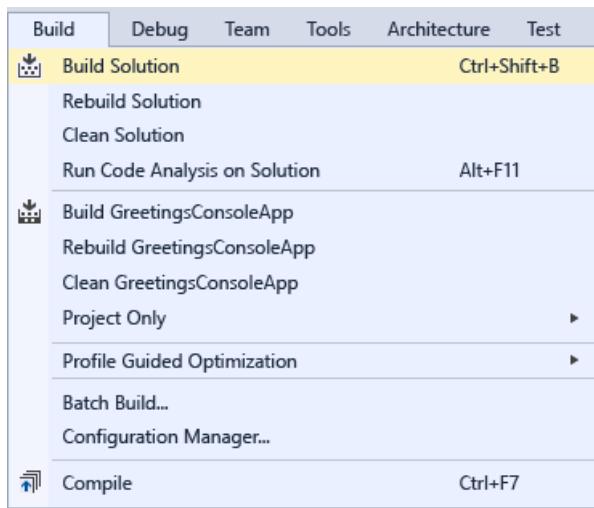
2. Choose the **Build** page.
3. In the **Configuration** list, choose **Release**.
4. In the **Platform** list, choose **x86**.
5. In the **Output path** box, specify a network path.

For example, you could specify \\myserver\builds.

IMPORTANT

A message box might appear, warning you that the network share that you've specified might not be a trusted location. If you trust the location that you've specified, choose the **OK** button in the message box.

6. On the **Standard toolbar**, set the Solution Configurations to **Release** and the Solution Platforms to **x86**.
7. Build the application.



The executable file is copied to the network path that you specified. Its path would be \\myserver\builds\FileName.exe.

Congratulations: you've successfully completed this walkthrough.

See also

[Walkthrough: Building a Project \(C++\)](#)

[ASP.NET Web Application Project Precompilation Overview](#)

[Walkthrough: Using MSBuild](#)

Building and Cleaning Projects and Solutions in Visual Studio

1/26/2018 • 2 min to read • [Edit Online](#)

By using the procedures in this topic, you can build, rebuild, or clean all or some of the projects or project items in a solution. For a step-by-step tutorial, see [Walkthrough: Building an Application](#).

NOTE

The UI in your edition of Visual Studio might differ from what this topic describes, depending on your active settings. To change your settings, for example to **General** or **Visual C++** settings, choose **Tools**, **Import and Export Settings**, and then choose **Reset all settings**.

To build, rebuild, or clean an entire solution

1. In **Solution Explorer**, choose or open the solution.
2. On the menu bar, choose **Build**, and then choose one of the following commands:
 - Choose **Build** or **Build Solution** to compile only those project files and components that have changed since the most recent build.

NOTE

The **Build** command becomes **Build Solution** when a solution includes more than one project.

- Choose **Rebuild Solution** to "clean" the solution and then build all project files and components.
- Choose **Clean Solution** to delete any intermediate and output files. With only the project and component files left, new instances of the intermediate and output files can then be built.

To build or rebuild a single project

1. In **Solution Explorer**, choose or open the project.
2. On the menu bar, choose **Build**, and then choose either **BuildProjectName** or **RebuildProjectName**.
 - Choose **BuildProjectName** to build only those project components that have changed since the most recent build.
 - Choose **RebuildProjectName** to "clean" the project and then build the project files and all project components.

To build only the startup project and its dependencies

1. On the menu bar, choose **Tools**, **Options**.
2. In the **Options** dialog box, expand the **Projects and Solutions** node, and then choose the **Build and Run** page.

The **Build and Run, Projects and Solutions, Options** dialog box opens.
3. Select the **Only build startup projects and dependencies on Run** check box.

When this check box is selected, only the current startup project and its dependencies are built when you

perform either of the following steps:

- On the menu bar, choose **Debug, Start** (F5).
- On the menu bar, choose **Build, Build Solution** (CTRL+SHIFT+B).

When this check box is cleared, all projects, their dependencies, and the solution files are built when you run either of the preceding commands. By default, this check box is cleared.

To build only the selected Visual C++ project

1. Choose a Visual C++ project, and then, on the menu bar, choose **Build, Project Only**, and one of the following commands:

- **Build Only** *ProjectName*
- **Rebuild Only** *ProjectName*
- **Clean Only** *ProjectName*
- **Link Only** *ProjectName*

These commands apply only to the Visual C++ project that you chose, without building, rebuilding, cleaning, or linking any project dependencies or solution files. Depending on your version of Visual Studio, the **Project Only** submenu might contain more commands.

To compile multiple C++ project items

1. In **Solution Explorer**, choose multiple files that have can be compiled actions, open the shortcut menu for one of those files, and then choose **Compile**.

If the files have dependencies, the files will be compiled in dependency order. The compile operation will fail if the files require a precompiled header that isn't available when you compile. The compile operation uses the current active solution configuration.

To stop a build

1. Perform either of the following steps:

- On the menu bar, choose **Build, Cancel**.
- Choose the Ctrl + Break keys.

See also

[How to: View, Save, and Configure Build Log Files](#)

[Obtaining Build Logs](#)

[Compiling and Building](#)

[Understanding Build Configurations](#)

[How to: Set debug and release configurations](#)

[C/C++ Building Reference](#)

[Devenv Command Line Switches](#)

[Solutions and Projects](#)

How to: Change the Build Output Directory

1/26/2018 • 1 min to read • [Edit Online](#)

You can specify the location of output on a per-configuration basis (for debug, release, or both) generated by your project.

NOTE

If you have a **Setup** project see the note at the end of this article.

Changing the Build Output directory

To change the build output directory

1. On the menu bar, choose **Project**, *Appname Properties*. You can also right-click the project node in the **Solution Explorer** and select **Properties**.
2. If you have a Visual Basic project, select the **Compile** tab. If you have a C# project, select the **Build** tab. If you have a C++ project or a JavaScript project, select the **General** tab.
3. In the configuration drop-down at the top, choose the configuration whose output file location you want to change (debug, release, or all).

Find the output path entry (**Build output path** in Visual Basic, **Output Directory** in Visual C++, **Output Path** in JavaScript and C#). Specify a new build output directory relative to the project directory.

NOTE

In a Setup Project, the **Output file name** box changes only the location of the Setup.exe file, not the location of the project files. For more information, see **Build**, **Configuration Properties**, **Deployment Project Properties Dialog Box**.

See Also

[Build Page, Project Designer \(C#\)](#)

[General Property Page \(Project\)](#)

[Compiling and Building](#)

How to: Build to a Common Output Directory

12/22/2017 • 1 min to read • [Edit Online](#)

By default, Visual Studio builds each project in a solution in its own folder inside the solution. You can change the build output paths of your projects to force all outputs to be placed in the same folder.

To place all solution outputs in a common directory

1. Click on one project in the solution.
2. On the **Project** menu, click **Properties**.
3. Depending on the type of project, click on either the **Compile** tab or the **Build** tab, and set the **Output path** to a folder to use for all projects in the solution.
4. Repeat steps 1-3 for all projects in the solution.

See Also

[Compiling and Building](#)

[How to: Change the Build Output Directory](#)

Specifying Custom Build Events in Visual Studio

1/26/2018 • 1 min to read • [Edit Online](#)

By specifying a custom build event, you can automatically run commands before a build starts or after it finishes. For example, you can run a .bat file before a build starts or copy new files to a folder after the build is complete. Build events run only if the build successfully reaches those points in the build process.

For specific information about the programming language that you're using, see the following topics:

- Visual Basic--[How to: Specify Build Events \(Visual Basic\)](#).
- C# and F#--[How to: Specify Build Events \(C#\)](#).
- Visual C++--[Specifying Build Events](#).

Syntax

Build events follow the same syntax as DOS commands, but you can use macros to create build events more easily.

For a list of available macros, see [Pre-build Event/Post-build Event Command Line Dialog Box](#).

For best results, follow these formatting tips:

- Add a `call` statement before all build events that run .bat files.

Example: `call C:\MyFile.bat`

Example: `call C:\MyFile.bat call C:\MyFile2.bat`

- Enclose file paths in quotation marks.

Example (for Windows 8): `"%ProgramFiles(x86)%\Microsoft SDKs\Windows\v8.0A\Bin\NETFX 4.0 Tools\gacutil.exe" -if "$(TargetPath)"`

- Separate multiple commands by using line breaks.
- Include wildcards as needed.

Example: `for %I in (*.txt *.doc *.html) do copy %I c:\mydirectory\`

NOTE

`%I` in the code above should be `%%I` in batch scripts.

See Also

[Compiling and Building](#)

[Pre-build Event/Post-build Event Command Line Dialog Box](#)

[MSBuild Special Characters](#)

[Walkthrough: Building an Application](#)

How to: Set Multiple Startup Projects

12/22/2017 • 1 min to read • [Edit Online](#)

Visual Studio allows you to specify how more than one project is run when you start the debugger.

To set multiple startup projects

1. In the **Solution Explorer**, select the solution (the very top node).
2. Choose the solution node's context (right-click) menu and then choose **Properties**. The **Solution Property Pages** dialog box appears.
3. Expand the **Common Properties** node, and choose **Startup Project**.
4. Choose the **Multiple Startup Projects** option and set the appropriate actions.

See Also

[Compiling and Building](#)

[Creating Solutions and Projects](#)

[Managing Project and Solution Properties](#)

How to: Create and Remove Project Dependencies

12/22/2017 • 1 min to read • [Edit Online](#)

When building a solution that contains multiple projects, it can be necessary to build certain projects first, to generate code used by other projects. When a project consumes executable code generated by another project, the project that generates the code is referred to as a project dependency of the project that consumes the code. Such dependency relationships can be defined in the **Project Dependencies** Dialog Box.

To assign dependencies to projects

1. In Solution Explorer, select a project.
2. On the **Project** menu, choose **Project Dependencies**.

The **Project Dependencies** dialog box opens.

NOTE

The **Project Dependencies** option is only available in a solution with more than one project.

3. On the **Dependencies** tab, select a project from the **Project** drop-down menu.
4. In the **Depends on** field, select the check box of any other project that must build before this project does.

Your solution must consist of more than one project before you can create project dependencies.

To remove dependencies from projects

1. In Solution Explorer, select a project.
2. On the **Project** menu, choose **Project Dependencies**.

The **Project Dependencies** dialog box opens.

NOTE

The **Project Dependencies** option is only available in a solution with more than one project.

3. On the **Dependencies** tab, select a project from the **Project** drop-down menu.
4. In the **Depends on** field, clear the check boxes beside any other projects that are no longer dependencies of this project.

See Also

[Building and Cleaning Projects and Solutions in Visual Studio](#)

[Compiling and Building](#)

[Understanding Build Configurations](#)

[Managing Project and Solution Properties](#)

How to: View, Save, and Configure Build Log Files

12/22/2017 • 2 min to read • [Edit Online](#)

After you build a project in the Visual Studio IDE, you can view information about that build in the **Output** window. By using this information, you can, for example, troubleshoot a build failure. For C++ projects, you can also view the same information in a .txt file that's created and saved automatically. For managed-code projects, you can copy and paste the information from the **Output** window into a .txt file and save it yourself. You can also use the IDE to specify what kinds of information you want to view about each build.

If you build any kind of project by using MSBuild, you can create a .txt file to save information about the build. For more information, see [Obtaining Build Logs](#).

To view the build log file for a C++ project

1. In **Windows Explorer** or **File Explorer**, open the following file: \...\Visual Studio Version\Projects\ProjectName\ProjectName\Debug\ProjectName.txt

To create a build log file for a managed-code project

1. On the menu bar, choose **Build**, **Build Solution**.
2. In the **Output** window, highlight the information from the build, and then copy it to the Clipboard.
3. Open a text editor, such as Notepad, paste the information into the file, and then save it.

To change the amount of information included in the build log

1. On the menu bar, choose **Tools**, **Options**.
2. On the **Projects and Solutions** page, choose the **Build and Run** page.
3. In the **MSBuild project build output verbosity** list, choose one of the following values, and then choose the **OK** button.

| VERBOSITY LEVEL | DESCRIPTION |
|-----------------|---|
| Quiet | Displays a summary of the build only. |
| Minimal | Displays a summary of the build and errors, warnings, and messages that are categorized as highly important. |
| Normal | Displays a summary of the build; errors, warnings, and messages that are categorized as highly important; and the main steps of the build. You'll use this level of detail most frequently. |
| Detailed | Displays a summary of the build; errors, warnings, and messages that are categorized as highly important; all of the steps of the build; and messages that are categorized as of normal importance. |
| Diagnostic | Displays all data that's available for the build. You can use this level of detail to help debug issues with custom build scripts and other build issues. |

For more information, see [Options Dialog Box, Projects and Solutions, Build and Run](#) and [LoggerVerbosity](#).

IMPORTANT

You must rebuild the project for your changes to take effect in the **Output** window (all projects) and the *ProjectName.txt* file (C++ projects only).

See Also

[Obtaining Build Logs](#)

[Building and Cleaning Projects and Solutions in Visual Studio](#)

[Compiling and Building](#)

How to: Exclude Projects from a Build

12/22/2017 • 1 min to read • [Edit Online](#)

You can build a solution without building all projects that it contains. For example, you might exclude a project that breaks the build. You could then build the project after you investigate and address the issues.

You can exclude a project by taking the following approaches:

- Removing it temporarily from the active solution configuration.
- Creating a solution configuration that doesn't include the project.

For more information, see [Understanding Build Configurations](#).

To temporarily remove a project from the active solution configuration

1. On the menu bar, choose **Build, Configuration Manager**.
2. In the **Project contexts** table, locate the project you want to exclude from the build.
3. In the **Build** column for the project, clear the check box.
4. Choose the **Close** button, and then rebuild the solution.

To create a solution configuration that excludes a project

1. On the menu bar, choose **Build, Configuration Manager**.
2. In the **Active solution configuration** list, choose **<New>**.
3. In the **Name** box, enter a name for the solution configuration.
4. In the **Copy settings from** list, choose the solution configuration on which you want to base the new configuration (for example, **Debug**), and then choose the **OK** button.
5. In the **Configuration Manager** dialog box, clear the check box in the **Build** column for the project that you want to exclude, and then choose the **Close** button.
6. On the **Standard** toolbar, verify that the new solution configuration is the active configuration in the **Solution Configurations** box.
7. On the menu bar, choose **Build, Rebuild Solution**.

See Also

[Understanding Build Configurations](#)

[How to: Create and Edit Configurations](#)

[How to: Build Multiple Configurations Simultaneously](#)

How to: Suppress compiler warnings

1/29/2018 • 3 min to read • [Edit Online](#)

You can declutter a build log by filtering out one or more kinds of compiler warnings. For example, you might want to review only some of the output that's generated when you set the build log verbosity to Normal, Detailed, or Diagnostic. For more information about verbosity, see [How to: View, save, and configure build log files](#).

Suppressing specific warnings for Visual C# or F#

Use the **Build** property page to suppress specific warnings for C# and F# projects.

1. In **Solution Explorer**, choose the project in which you want to suppress warnings.
2. On the menu bar, choose **View > Property Pages**.
3. Choose the **Build** page.
4. In the **Suppress warnings** box, specify the error codes of the warnings that you want to suppress, separated by semicolons.
5. Rebuild the solution.

Suppressing specific warnings for Visual C++

Use the **Configuration Properties** property page to suppress specific warnings for C++ projects.

1. In **Solution Explorer**, choose the project or source file in which you want to suppress warnings.
2. On the menu bar, choose **View > Property Pages**.
3. Choose the **Configuration Properties** category, choose the **C/C++** category, and then choose the **Advanced** page.
4. Perform one of the following steps:
 - In the **Disable Specific Warnings** box, specify the error codes of the warnings that you want to suppress, separated by a semicolon.
 - In the **Disable Specific Warnings** box, choose **Edit** to display more options.
5. Choose the **OK** button, and then rebuild the solution.

Suppressing warnings for Visual Basic

You can hide specific compiler warnings for Visual Basic by editing the `.vbproj` file for the project. To suppress warnings by *category*, you can use the [Compile property page](#). For more information, see [Configuring warnings in Visual Basic](#).

To suppress specific warnings for Visual Basic

This example shows you how to edit the `.vbproj` file to suppress specific compiler warnings.

1. In **Solution Explorer**, choose the project in which you want to suppress warnings.
2. On the menu bar, choose **Project > Unload Project**.
3. In **Solution Explorer**, open the right-click or shortcut menu for the project, and then choose **Edit**

ProjectName.vbproj.

The XML project file opens in the code editor.

4. Locate the `<NoWarn>` element for the build configuration you're building with, and add one or more warning numbers as the value of the `<NoWarn>` element. If you specify multiple warning numbers, separate them with a comma.

The following example shows the `<NoWarn>` element for the *Debug* build configuration on an x86 platform, with two compiler warnings suppressed:

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|x86' ">
  <PlatformTarget>x86</PlatformTarget>
  <DebugSymbols>true</DebugSymbols>
  <DebugType>full</DebugType>
  <Optimize>false</Optimize>
  <OutputPath>bin\Debug\</OutputPath>
  <DefineDebug>true</DefineDebug>
  <DefineTrace>true</DefineTrace>
  <ErrorReport>prompt</ErrorReport>
  <NoWarn>40059,42024</NoWarn>
  <WarningLevel>1</WarningLevel>
</PropertyGroup>
```

NOTE

.NET Core projects do not contain build configuration property groups by default. To suppress warnings in a .NET Core project, add the build configuration section to the file manually. For example:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.0</TargetFramework>
    <RootNamespace>VBDotNetCore_1</RootNamespace>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <NoWarn>42016,41999,42017</NoWarn>
  </PropertyGroup>
</Project>
```

5. Save the changes to the *.vbproj* file.
6. On the menu bar, choose **Project** > **Reload Project**.
7. On the menu bar, choose **Build** > **Rebuild Solution**.

The **Output** window no longer shows the warnings that you specified.

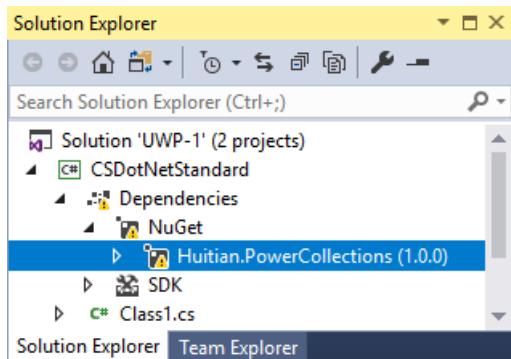
For more information, see the [/nowarn compiler option](#) for the Visual Basic command-line compiler.

Suppressing warnings for NuGet packages

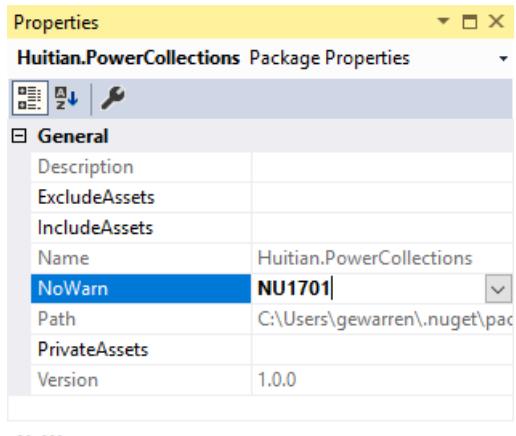
In some cases, you may want to suppress NuGet compiler warnings for a single NuGet package, instead of for an entire project. The warning serves a purpose, so you don't want to suppress it at the project level. For example, one of the NuGet warnings tells you that the package may not be fully compatible with your project. If you suppress it at the project level and later add an additional NuGet package, you would never know if it was producing the compatibility warning.

To suppress a specific warning for a single NuGet package

1. In **Solution Explorer**, select the NuGet package you want to suppress compiler warnings for.



2. From the right-click or context menu, select **Properties**.
3. In the **NoWarn** box of the package's properties, enter the warning number you want to suppress for this package. If you want to suppress more than one warning, use a comma to separate the warning numbers.



The warning disappears from **Solution Explorer** and the **Error List**.

See also

- [Walkthrough: Building an Application](#)
- [How to: View, Save, and Configure Build Log Files](#)
- [Compiling and Building](#)

Understanding Build Configurations

12/22/2017 • 4 min to read • [Edit Online](#)

You can store different configurations of solution and project properties to use in different kinds of builds. To create, select, modify, or delete a configuration, you can use the **Configuration Manager**. To open it, on the menu bar, choose **Build**, **Configuration Manager**, or just type **Configuration** in the **Quick Launch** box. You can also use the **Solution Configurations** list on the **Standard** toolbar to select a configuration or open the **Configuration Manager**.

NOTE

If you can't find solution configuration settings on the toolbar and can't access the **Configuration Manager**, Visual Basic development settings may be applied. For more information, see [How to: Manage Configurations with Visual Basic Developer Settings Applied](#).

By default, Debug and Release configurations are included in projects that are created by using Visual Studio templates. A Debug configuration supports the debugging of an app, and a Release configuration builds a version of the app that can be deployed. For more information, see [How to: Set Debug and Release Configurations](#). You can also create custom solution configurations and project configurations. For more information, see [How to: Create and Edit Configurations](#).

Solution Configurations

A solution configuration specifies how projects in the solution are to be built and deployed. To modify a solution configuration or define a new one, in the **Configuration Manager**, under **Active solution configuration**, choose **Edit** or **New**.

Each entry in the **Project contexts** box in a solution configuration represents a project in the solution. For every combination of **Active solution configuration** and **Active solution platform**, you can set how each project is used. (For more information about solution platforms, see [Understanding Build Platforms](#).)

NOTE

When you define a new solution configuration and select the **Create new project configurations** check box, Visual Studio automatically assigns the new configuration to all of the projects. Likewise, when you define a new solution platform and select the **Create new project platforms** check box, Visual Studio automatically assigns the new platform to all of the projects. Also, if you add a project that targets a new platform, Visual Studio adds that platform to the list of solution platforms and assigns it to all of the projects.

You can still modify the settings for each project.

The active solution configuration also provides context to the IDE. For example, if you're working on a project and the configuration specifies that it will be built for a mobile device, the **Toolbox** displays only items that can be used in a mobile device project.

Project Configurations

The configuration and platform that a project targets are used together to specify the properties to use when it's built. A project can have a different set of property definitions for each combination of configuration and platform. To modify the properties of a project, you can use its Property Pages. (In **Solution Explorer**, open the

shortcut menu for the project and then choose **Properties**.)

For each project configuration, you can define configuration-dependent properties as needed. For example, for a particular build, you can set which project items will be included, and what output files will be created, where they will be put, and how they will be optimized.

Project configurations can differ considerably. For example, the properties of one configuration might specify that its output file be optimized to occupy the minimum space, while another configuration might specify that its executable runs at the maximum speed.

Project configurations are stored by solution—not by user—so that they can be shared by a team.

Although project dependencies are configuration-independent, only the projects that are specified in the active solution configuration will be built.

How Visual Studio Assigns Project Configurations

When you define a new solution configuration and don't copy settings from an existing one, Visual Studio uses the following criteria to assign default project configurations. The criteria are evaluated in the order shown.

1. If a project has a configuration name (*<configuration name> <platform name>*) that exactly matches the name of the new solution configuration, that configuration is assigned. Configuration names are not case-sensitive.
2. If the project has a configuration name in which the configuration-name part matches the new solution configuration, that configuration is assigned, whether the platform portion matches or not.
3. If there is still no match, the first configuration that's listed in the project is assigned.

How Visual Studio Assigns Solution Configurations

When you create a project configuration (in the **Configuration Manager**, by choosing **New** on the drop-down menu in the **Configuration** column for that project) and select the **Create new solution configurations** check box, Visual Studio looks for a like-named solution configuration to build the project on each platform it supports. In some cases, Visual Studio renames existing solution configurations or defines new ones.

Visual Studio uses the following criteria to assign solution configurations.

- If a project configuration doesn't specify a platform or specifies just one platform, then a solution configuration whose name matches that of the new project configuration is either found or added. The default name of this solution configuration does not include a platform name; it takes the form *<project configuration name>*.
- If a project supports multiple platforms, a solution configuration is either found or added for each supported platform. The name of each solution configuration includes both the project configuration name and the platform name, and has the form *<project configuration name> <platform name>*.

See Also

[Walkthrough: Building an Application](#)

[Compiling and Building](#)

[Solutions and Projects](#)

[C/C++ Building Reference](#)

[Devenv Command Line Switches](#)

How to: Create and Edit Configurations

12/22/2017 • 3 min to read • [Edit Online](#)

You can create several build configurations for a solution. For example, you can configure a debug build that your testers can use to find and fix problems, and you can configure different kinds of builds that you can distribute to different customers.

> [!NOTE] > Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

Creating Build Configurations

You can use the **Configuration Manager** dialog box to select or modify existing build configurations, or you can create new ones.

To open the Configuration Manager dialog box

- In **Solution Explorer**, open the shortcut menu for the solution and then choose **Configuration Manager**.

NOTE

If the **Configuration Manager** command doesn't appear on the shortcut menu, look under the **Build** menu on the menu bar. If it doesn't appear there either, on the menu bar, choose **Tools, Options**, and then in the left pane of the **Options** dialog box, expand **Projects and Solutions, General**, and in the right pane, select the **Show advanced build configurations** check box.

In the **Configuration Manager** dialog box, you can use the **Active solution configuration** drop-down list to select a solution-wide build configuration, modify an existing one, or create a new configuration. You can use the **Active solution platform** drop-down list to select the platform that the configuration targets, modify an existing one, or add a new platform. The **Project contexts** pane lists the projects in the solution. For each project, you can select a project-specific configuration and platform, modify existing ones, or create a new configuration or add a new platform. You can also select check boxes that indicate whether each project is included when you use the solution-wide configuration to build or deploy the solution.

After you set up the configurations you want, you can set project properties that are appropriate for those configurations.

To set properties based on configurations

- In **Solution Explorer**, open the shortcut menu for a project and then choose **Properties**.

The **Property Pages** window opens.

You can set properties for your configurations. For example, for a Release configuration, you can specify that code is optimized when the solution is built, and for a Debug configuration, you can specify that the **DEBUG** conditional compilation symbol is included. For more information about property page settings, see [Managing Project and Solution Properties](#).

Creating and Modifying Project Configurations

To create a project configuration

1. Open the **Configuration Manager** dialog box.

2. Select a project in the **Project** column.
3. In the **Configuration** drop-down list for that project, choose **New**.

The **New Project Configuration** dialog box opens.

4. In the **Name** box, enter a name for the new configuration.
5. To use the property settings from an existing project configuration, in the **Copy settings from** drop-down list, choose a configuration.
6. To create a solution-wide configuration at the same time, select the **Create new solution configuration** check box.

To rename a project configuration

1. Open the **Configuration Manager** dialog box.
2. In the **Project** column, select the project that has the project configuration you want to rename.
3. In the **Configuration** drop-down list for that project, choose **Edit**.

The **Edit Project Configurations** dialog box opens.

4. Select the project configuration name you want to change.
5. Select **Rename**, and then enter a new name.

Creating and Modifying Solution-wide Build Configurations

To create a solution-wide build configuration

1. Open the **Configuration Manager** dialog box.
2. In the **Active solution configuration** drop-down list, choose **New**.

The **New Solution Configuration** dialog box opens.

3. In the **Name** text box, enter a name for the new configuration.
4. To use the settings from an existing solution configuration, in the **Copy settings from** drop-down list, choose a configuration.
5. If you want to create project configurations at the same time, select the **Create new project configurations** check box.

To rename a solution-wide build configuration

1. Open the **Configuration Manager** dialog box.
2. In the **Active solution configuration** drop-down list, choose **Edit**.

The **Edit Solution Configurations** dialog box opens.

3. Select the solution configuration name you want to change.
4. Select **Rename**, and then enter a new name.

To modify a solution-wide build configuration

1. Open the **Configuration Manager** dialog box.
2. In the **Active solution configuration** drop-down list, select the configuration you want.
3. In the **Project contexts** pane, for every project, select the **Configuration** and **Platform** you want, and select whether to **Build** it and whether to **Deploy** it..

See Also

[Understanding Build Configurations](#)

[Building and Cleaning Projects and Solutions in Visual Studio](#)

[Managing Project and Solution Properties](#)

How to: Manage Build Configurations with Visual Basic Developer Settings Applied

12/22/2017 • 1 min to read • [Edit Online](#)

By default, all advanced build configuration options are hidden with Visual Basic Developer settings applied. This topic explains how to manually enable these settings.

Enabling advanced build configurations

By default, the Visual Basic Developer settings hide the option to open the **Configuration Manager** dialog box and the **Configuration** and **Platform** lists in the **Project Designer**.

To enable advanced build configurations

1. On the **Tools** menu, click **Options**.
2. Expand **Projects and Solutions**, and click **General**.

NOTE

The **General** node is visible even if the **Show all settings** option is unchecked. If you want to see every option available, click **Show all settings**.

3. Click **Show advanced build configurations**.
4. Click **OK**.

On the **Build** menu, **Configuration Manager** is now available, and the **Configuration** and **Platform** lists are visible in the Project Designer.

See Also

[Understanding Build Configurations](#)

[Compiling and Building](#)

How to: Build Multiple Configurations Simultaneously

12/22/2017 • 1 min to read • [Edit Online](#)

You can build most types of projects with multiple, or even all, of their build configurations at the same time by using the **Batch Build** dialog box. However, you can't build the following types of projects in multiple build configurations at the same time:

1. Windows 8.x Store apps built for Windows using JavaScript.
2. All Visual Basic projects.

For more information about build configurations, see [Understanding Build Configurations](#).

To build a project in multiple build configurations

1. On the menu bar, choose **Build**, **Batch Build**.
2. In the **Build** column, select the check boxes for the configurations in which you want to build a project.

TIP

To edit or create a build configuration for a solution, choose **Build**, **Configuration Manager** on the menu bar to open the **Configuration Manager** dialog box. After you have edited a build configuration for a solution, choose the **Rebuild** button in the **Batch Build** dialog box to update all build configurations for the projects in the solution.

3. Choose the **Build** or **Rebuild** buttons to build the project with the configurations that you specified.

See Also

[How to: Create and Edit Configurations](#)

[Understanding Build Configurations](#)

[Building Multiple Projects in Parallel](#)

Understanding Build Platforms

12/22/2017 • 1 min to read • [Edit Online](#)

You can store different versions of solution and project properties that apply to different target platforms. For example, you can create a Debug configuration that targets an x86 platform and a Debug configuration that targets an x64 platform. You can quickly change the active platform so that you can easily build multiple configurations.

In This Section

[How to: Configure Projects to Target Platforms](#)

Explains how to configure a project to target a specific platform.

[How to: Configure Projects to Target Multiple Platforms](#)

Explains how to configure a project to target multiple platforms.

See Also

[Walkthrough: Building an Application](#)

[Building and Cleaning Projects and Solutions in Visual Studio](#)

[Compiling and Building](#)

How to: Configure Projects to Target Platforms

12/22/2017 • 1 min to read • [Edit Online](#)

Visual Studio enables you to set up your applications to target different platforms, including 64-bit platforms. For more information on 64-bit platform support in Visual Studio, see [64-bit Applications](#).

Targeting Platforms with the Configuration Manager

The **Configuration Manager** provides a way for you to quickly add a new platform to target with your project. If you select one of the platforms included with Visual Studio, the properties for your project are modified to build your project for the selected platform.

To configure a project to target a 64-bit platform

1. On the menu bar, choose **Build, Configuration Manager**.
2. In the **Active solution platform** list, choose a 64-bit platform for the solution to target, and then choose the **Close** button.
 - a. If the platform that you want doesn't appear in the **Active solution platform** list, choose **New**.
The **New Solution Platform** dialog box appears.
 - b. In the **Type or select the new platform** list, choose **x64**.

NOTE

If you give your configuration a new name, you may have to modify the settings in the **Project Designer** to target the correct platform.

- c. If you want to copy the settings from a current platform configuration, choose it, and then choose the **OK** button.

The properties for all projects that target the 64-bit platform are updated, and the next build of the project will be optimized for 64-bit platforms.

Targeting Platforms in the Project Designer

The Project Designer also provides a way to target different platforms with your project. If selecting one of the platforms included in the list in the **New Solution Platform** dialog box does not work for your solution, you can create a custom configuration name and modify the settings in the **Project Designer** to target the correct platform.

Performing this task varies based on the programming language you are using. See the following links for more information:

- For Visual Basic projects, see [/platform \(Visual Basic\)](#).
- For Visual C# projects, see [Build Page, Project Designer \(C#\)](#).
- For Visual C++ projects, see [/clr \(Common Language Runtime Compilation\)](#).

See Also

[Understanding Build Platforms](#)

[/platform \(C# Compiler Options\)](#)

[64-bit Applications](#)

[Visual Studio IDE 64-Bit Support](#)

How to: Configure Projects to Target Multiple Platforms

12/22/2017 • 2 min to read • [Edit Online](#)

Visual Studio provides a way for a solution to target several different CPU architectures, or platforms, at once. The properties to set these are accessed through the **Configuration Manager** dialog box.

Targeting a Platform

The **Configuration Manager** dialog box allows you to create and set solution-level and project-level configurations and platforms. Each combination of solution-level configurations and targets can have a unique set of properties associated with it, allowing you to easily switch between, for example, a Release configuration that targets an x64 platform, a Release configuration that targets an x86 platform, and a Debug configuration that targets an x86 platform.

To set your configuration to target a different platform

1. On the **Build** menu, click **Configuration Manager**.
2. In the **Active solution platform box**, select the platform you want your solution to target, or select **<New>** to create a new platform. Visual Studio will compile your application to target the platform that is set as the active platform in the **Configuration Manager** dialog box.

Removing a Platform

If you realize that you have no need for a platform, you can remove it using the Configuration Manager dialog box. This will remove all solution and project settings that you configured for that combination of configuration and target.

To remove a platform

1. On the **Build** menu, click **Configuration Manager**.
2. In the **Active solution platform box**, select **<Edit>**. The **Edit Solution Platforms** dialog box opens.
3. Click the platform you want to remove, and click **Remove**.

Targeting Multiple Platforms with One Solution

Because you can change the settings based on the combination of configuration and platform settings, you can set up a solution that can target more than one platform.

To target multiple platforms

1. Use the **Configuration Manager** to add at least two target platforms for the solution.
2. Select the platform you want to target from the **Active solution platform** list.
3. Build the solution.

To build multiple solution configurations at once

1. Use the **Configuration Manager** to add at least two target platforms for the solution.
2. Use the **Batch Build** window to build several solution configurations at once.

It is possible to have a solution-level platform set to, for example, x64, and have no projects within that

solution targeting the same platform. It is also possible to have multiple projects in your solution, each targeting different platforms. It is recommended that if you have one of these situations, you create a new configuration with a descriptive name to avoid confusion.

See Also

[How to: Create and Edit Configurations](#)

[Understanding Build Configurations](#)

[Building and Cleaning Projects and Solutions in Visual Studio](#)

MSBuild

2/8/2018 • 9 min to read • [Edit Online](#)

The Microsoft Build Engine is a platform for building applications. This engine, which is also known as MSBuild, provides an XML schema for a project file that controls how the build platform processes and builds software. Visual Studio uses MSBuild, but it doesn't depend on Visual Studio. By invoking msbuild.exe on your project or solution file, you can orchestrate and build products in environments where Visual Studio isn't installed.

Visual Studio uses MSBuild to load and build managed projects. The project files in Visual Studio (.csproj,.vbproj, .vcxproj, and others) contain MSBuild XML code that executes when you build a project by using the IDE. Visual Studio projects import all the necessary settings and build processes to do typical development work, but you can extend or modify them from within Visual Studio or by using an XML editor.

For information about MSBuild for C++, see [MSBuild \(Visual C++\)](#).

The following examples illustrate when you might run builds by using an MSBuild command line instead of the Visual Studio IDE.

- Visual Studio isn't installed.
- You want to use the 64-bit version of MSBuild. This version of MSBuild is usually unnecessary, but it allows MSBuild to access more memory.
- You want to run a build in multiple processes. However, you can use the IDE to achieve the same result on projects in C++ and C#.
- You want to modify the build system. For example, you might want to enable the following actions:
 - Preprocess files before they reach the compiler.
 - Copy the build outputs to a different place.
 - Create compressed files from build outputs.
 - Do a post-processing step. For example, you might want to stamp an assembly with a different version.

You can write code in the Visual Studio IDE but run builds by using MSBuild. As another alternative, you can build code in the IDE on a development computer but use an MSBuild command line to build code that's integrated from multiple developers.

NOTE

You can use Team Foundation Build to automatically compile, test, and deploy your application. Your build system can automatically run builds when developers check in code (for example, as part of a Continuous Integration strategy) or according to a schedule (for example, a nightly Build Verification Test build). Team Foundation Build compiles your code by using MSBuild. For more information, see [Build and release](#).

This topic provides an overview of MSBuild. For an introductory tutorial, see [Walkthrough: Using MSBuild](#).

In this topic

- [Using MSBuild at a Command Prompt](#)
- [Project File](#)

- [Properties](#)
- [Items](#)
- [Tasks](#)
- [Targets](#)
- [Build Logs](#)
- [Using MSBuild in Visual Studio](#)
- [Multitargeting](#)

Using MSBuild at a Command Prompt

To run MSBuild at a command prompt, pass a project file to MSBuild.exe, together with the appropriate command-line options. Command-line options let you set properties, execute specific targets, and set other options that control the build process. For example, you would use the following command-line syntax to build the file

`MyProj.proj` with the `Configuration` property set to `Debug`.

```
MSBuild.exe MyProj.proj /property:Configuration=Debug
```

For more information about MSBuild command-line options, see [Command-Line Reference](#).

IMPORTANT

Before you download a project, determine the trustworthiness of the code.

Project File

MSBuild uses an XML-based project file format that's straightforward and extensible. The MSBuild project file format lets developers describe the items that are to be built, and also how they are to be built for different operating systems and configurations. In addition, the project file format lets developers author reusable build rules that can be factored into separate files so that builds can be performed consistently across different projects in the product.

The following sections describe some of the basic elements of the MSBuild project file format. For a tutorial about how to create a basic project file, see [Walkthrough: Creating an MSBuild Project File from Scratch](#).

Properties

Properties represent key/value pairs that can be used to configure builds. Properties are declared by creating an element that has the name of the property as a child of a `PropertyGroup` element. For example, the following code creates a property named `BuildDir` that has a value of `Build`.

```
<PropertyGroup>
  <BuildDir>Build</BuildDir>
</PropertyGroup>
```

You can define a property conditionally by placing a `Condition` attribute in the element. The contents of conditional elements are ignored unless the condition evaluates to `true`. In the following example, the `Configuration` element is defined if it hasn't yet been defined.

```
<Configuration Condition="$(Configuration) == ''">Debug</Configuration>
```

Properties can be referenced throughout the project file by using the syntax `$(PropertyName)`. For example, you can reference the properties in the previous examples by using `$(BuildDir)` and `$(Configuration)`.

For more information about properties, see [MSBuild Properties](#).

Items

Items are inputs into the build system and typically represent files. Items are grouped into item types, based on user-defined item names. These item types can be used as parameters for tasks, which use the individual items to perform the steps of the build process.

Items are declared in the project file by creating an element that has the name of the item type as a child of an [ItemGroup](#) element. For example, the following code creates an item type named `Compile`, which includes two files.

```
<ItemGroup>
  <Compile Include = "file1.cs"/>
  <Compile Include = "file2.cs"/>
</ItemGroup>
```

Item types can be referenced throughout the project file by using the syntax `@(ItemType)`. For example, the item type in the example would be referenced by using `@(Compile)`.

In MSBuild, element and attribute names are case-sensitive. However, property, item, and metadata names are not. The following example creates the item type `Compile`, `comPile`, or any other case variation, and gives the item type the value "one.cs;two.cs".

```
<ItemGroup>
  <Compile Include="one.cs" />
  <comPile Include="two.cs" />
</ItemGroup>
```

Items can be declared by using wildcard characters and may contain additional metadata for more advanced build scenarios. For more information about items, see [Items](#).

Tasks

Tasks are units of executable code that MSBuild projects use to perform build operations. For example, a task might compile input files or run an external tool. Tasks can be reused, and they can be shared by different developers in different projects.

The execution logic of a task is written in managed code and mapped to MSBuild by using the [UsingTask](#) element. You can write your own task by authoring a managed type that implements the [ITask](#) interface. For more information about how to write tasks, see [Task Writing](#).

MSBuild includes common tasks that you can modify to suit your requirements. Examples are [Copy](#), which copies files, [MakeDir](#), which creates directories, and [Csc](#), which compiles Visual C# source code files. For a list of available tasks together with usage information, see [Task Reference](#).

A task is executed in an MSBuild project file by creating an element that has the name of the task as a child of a [Target](#) element. Tasks typically accept parameters, which are passed as attributes of the element. Both MSBuild properties and items can be used as parameters. For example, the following code calls the [MakeDir](#) task and passes it the value of the `BuildDir` property that was declared in the earlier example.

```
<Target Name="MakeBuildDirectory">
  <MakeDir Directories="$(BuildDir)" />
</Target>
```

For more information about tasks, see [Tasks](#).

Targets

Targets group tasks together in a particular order and expose sections of the project file as entry points into the build process. Targets are often grouped into logical sections to increase readability and to allow for expansion. Breaking the build steps into targets lets you call one piece of the build process from other targets without copying that section of code into every target. For example, if several entry points into the build process require references to be built, you can create a target that builds references and then run that target from every entry point where it's required.

Targets are declared in the project file by using the [Target](#) element. For example, the following code creates a target named `Compile`, which then calls the [Csc](#) task that has the item list that was declared in the earlier example.

```
<Target Name="Compile">
  <Csc Sources="@(<Compile>)" />
</Target>
```

In more advanced scenarios, targets can be used to describe relationships among one another and perform dependency analysis so that whole sections of the build process can be skipped if that target is up-to-date. For more information about targets, see [Targets](#).

Build Logs

You can log build errors, warnings, and messages to the console or another output device. For more information, see [Obtaining Build Logs](#) and [Logging in MSBuild](#).

Using MSBuild in Visual Studio

Visual Studio uses the MSBuild project file format to store build information about managed projects. Project settings that are added or changed by using the Visual Studio interface are reflected in the `.*proj` file that's generated for every project. Visual Studio uses a hosted instance of MSBuild to build managed projects. This means that a managed project can be built in Visual Studio or at a command prompt (even if Visual Studio isn't installed), and the results will be identical.

For a tutorial about how to use MSBuild in Visual Studio, see [Walkthrough: Using MSBuild](#).

Multitargeting

By using Visual Studio, you can compile an application to run on any one of several versions of the .NET Framework. For example, you can compile an application to run on the .NET Framework 2.0 on a 32-bit platform, and you can compile the same application to run on the .NET Framework 4.5 on a 64-bit platform. The ability to compile to more than one framework is named multitargeting.

These are some of the benefits of multitargeting:

- You can develop applications that target earlier versions of the .NET Framework, for example, versions 2.0, 3.0, and 3.5.
- You can target frameworks other than the .NET Framework, for example, Silverlight.
- You can target a *framework profile*, which is a predefined subset of a target framework.
- If a service pack for the current version of the .NET Framework is released, you could target it.
- Multitargeting guarantees that an application uses only the functionality that's available in the target framework and platform.

For more information, see [Multitargeting](#).

Related Topics

| TITLE | DESCRIPTION |
|--|---|
| Walkthrough: Creating an MSBuild Project File from Scratch | Shows how to create a basic project file incrementally, by using only a text editor. |
| Walkthrough: Using MSBuild | Introduces the building blocks of MSBuild and shows how to write, manipulate, and debug MSBuild projects without closing the Visual Studio IDE. |
| MSBuild Concepts | Presents the four building blocks of MSBuild: properties, items, targets, and tasks. |
| Items | Describes the general concepts behind the MSBuild file format and how the pieces fit together. |
| MSBuild Properties | Introduces properties and property collections. Properties are key/value pairs that can be used to configure builds. |
| Targets | Explains how to group tasks together in a particular order and enable sections of the build process to be called on the command line. |
| Tasks | Shows how to create a unit of executable code that can be used by MSBuild to perform atomic build operations. |
| Conditions | Discusses how to use the <code>Condition</code> attribute in an MSBuild element. |
| Advanced Concepts | Presents batching, performing transforms, multitargeting, and other advanced techniques. |
| Logging in MSBuild | Describes how to log build events, messages, and errors. |
| Additional Resources | Lists community and support resources for more information about MSBuild. |

Reference

[MSBuild Reference](#)

Links to topics that contain reference information.

[Glossary](#) Defines common MSBuild terms.

How to: Specify Build Events (Visual Basic)

1/26/2018 • 4 min to read • [Edit Online](#)

Build events in Visual Basic can be used to run scripts, macros, or other actions as a part of the compilation process. Pre-build events occur before compilation; post-build events occur after compilation.

Build events are specified in the **Build Events** dialog box, available from the **Compile** page of the **Project Designer**.

NOTE

Visual Basic Express does not support entry of build events. This is supported only in the full Visual Studio product.

How to Specify Pre-Build and Post-Build Events

To specify a build event

1. With a project selected in **Solution Explorer**, on the **Project** menu, click **Properties**.
2. Click the **Compile** tab.
3. Click the **Build Events** button to open the **Build Events** dialog box.
4. Enter the command-line arguments for your pre-build or post-build action, and then click **OK**.

NOTE

Add a `call` statement before all post-build commands that run .bat files. For example, `call C:\MyFile.bat` or `call C:\MyFile.bat call C:\MyFile2.bat`.

NOTE

If your pre-build or post-build event does not complete successfully, you can terminate the build by having your event action exit with a code other than zero (0), which indicates a successful action.

Example: How to Change Manifest Information Using a Post-Build Event

The following procedure shows how to set the minimum operating system version in the application manifest using an .exe command called from a post-build event (the .exe.manifest file in the project directory). The minimum operating system version is a four-part number such as 4.10.0.0. To do this, the command will change the `<dependentOS>` section of the manifest:

```
<dependentOS>
  <osVersionInfo>
    <os majorVersion="4" minorVersion="10" buildNumber="0" servicePackMajor="0" />
  </osVersionInfo>
</dependentOS>
```

To create an .exe command to change the application manifest

1. Create a console application for the command. From the **File** menu, click **New**, and then click **Project**.

2. In the **New Project** dialog box, in the **Visual Basic** node, select **Windows** and then the **Console Application** template. Name the project `changeOSVersionVB`.

3. In Module1.vb, add the following line to the other `Imports` statements at the top of the file:

```
Imports System.Xml
```

4. Add the following code in `Sub Main`:

```
Sub Main()
    Dim applicationManifestPath As String
    applicationManifestPath = My.Application.CommandLineArgs(0)
    Console.WriteLine("Application Manifest Path: " & applicationManifestPath.ToString)

    'Get version name
    Dim osVersion As Version
    If My.Application.CommandLineArgs.Count >= 2 Then
        osVersion = New Version(My.Application.CommandLineArgs(1).ToString)
    Else
        Throw New ArgumentException("OS Version not specified.")
    End If
    Console.WriteLine("Desired OS Version: " & osVersion.ToString())

    Dim document As XmlDocument
    Dim namespaceManager As XmlNamespaceManager
    namespaceManager = New XmlNamespaceManager(New NameTable())
    With namespaceManager
        .AddNamespace("asmv1", "urn:schemas-microsoft-com:asm.v1")
        .AddNamespace("asmv2", "urn:schemas-microsoft-com:asm.v2")
    End With

    document = New XmlDocument()
    document.Load(applicationManifestPath)

    Dim baseXPath As String
    baseXPath = "/asmv1:assembly/asmv2:dependency/asmv2:dependentOS/asmv2:osVersionInfo/asmv2:os"

    'Change minimum required OS Version.
    Dim node As XmlNode
    node = document.SelectSingleNode(baseXPath, namespaceManager)
    node.Attributes("majorVersion").Value = osVersion.Major.ToString()
    node.Attributes("minorVersion").Value = osVersion.Minor.ToString()
    node.Attributes("buildNumber").Value = osVersion.Build.ToString()
    node.Attributes("servicePackMajor").Value = osVersion.Revision.ToString()

    document.Save(applicationManifestPath)
End Sub
```

The command takes two arguments. The first argument is the path to the application manifest (that is, the folder in which the build process creates the manifest, typically `Projectname.publish`). The second argument is the new operating system version.

5. On the **Build** menu, click **Build Solution**.

6. Copy the .exe file to a directory such as `C:\TEMP\ChangeOSVersionVB.exe`.

Next, invoke this command in a post-build event to change the application manifest.

To invoke a post-build event to change the application manifest

1. Create a Windows application for the project to be published. From the **File** menu, click **New**, and then click **Project**.

2. In the **New Project** dialog box, in the **Visual Basic** node, select **Windows Classic Desktop** and then the **Windows Forms App** template. Name the project `VBWinApp`.
3. With the project selected in **Solution Explorer**, on the **Project** menu, click **Properties**.
4. In the Project Designer, go to the **Publish** page and set **Publishing location** to `C:\TEMP\`.
5. Publish the project by clicking **Publish Now**.

The manifest file will be built and put in `C:\TEMP\VBWinApp_1_0_0_0\VBWinApp.exe.manifest`. To view the manifest, right-click the file and click **Open with**, then click **Select the program from a list**, and then click **Notepad**.

Search in the file for the `<osVersionInfo>` element. For example, the version might be:

```
<os majorVersion="4" minorVersion="10" buildNumber="0" servicePackMajor="0" />
```

6. In the Project Designer, go to the **Compile** tab and click the **Build Events** button to open the **Build Events** dialog box.
7. In the **Post-build Event Command Line** box, enter the following command:

```
C:\TEMP\ChangeOSVersionVB.exe "$(TargetPath).manifest" 5.1.2600.0
```

When you build the project, this command will change the minimum operating system version in the application manifest to 5.1.2600.0.

The `$(TargetPath)` macro expresses the full path for the executable being created. Therefore, `$(TargetPath).manifest` will specify the application manifest created in the bin directory. Publishing will copy this manifest to the publishing location that you set earlier.

8. Publish the project again. Go to the **Publish** page and click **Publish Now**.

View the manifest again. To view the manifest, go to the publish directory, right-click the file and click **Open with** and then **Select the program from a list**, and then click **Notepad**.

The version should now read:

```
<os majorVersion="5" minorVersion="1" buildNumber="2600" servicePackMajor="0" />
```

See also

- [Compile Page, Project Designer \(Visual Basic\)](#)
- [Publish Page, Project Designer](#)
- [Pre-build Event/Post-build Event Command Line Dialog Box](#)
- [How to: Specify Build Events \(C#\)](#)

How to: Specify Build Events (C#)

12/22/2017 • 5 min to read • [Edit Online](#)

Use build events to specify commands that run before the build starts or after the build finishes. Build events are executed only if the build successfully reaches those points in the build process.

When a project is built, pre-build events are added to a file that is named PreBuildEvent.bat and post-build events are added to a file that is named PostBuildEvent.bat. If you want to ensure error checking, add your own error-checking commands to the build steps.

> [!NOTE] > Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

How to Specify Pre-Build and Post-Build Events

To specify a build event

1. In **Solution Explorer**, select the project for which you want to specify the build event.
2. On the **Project** menu, click **Properties**.
3. Select the **Build Events** tab.
4. In the **Pre-build event command line** box, specify the syntax of the build event.

NOTE

Pre-build events do not run if the project is up to date and no build is triggered.

5. In the **Post-build event command line** box, specify the syntax of the build event.

NOTE

Add a `call` statement before all post-build commands that run .bat files. For example, `call C:\MyFile.bat` or `call C:\MyFile.bat call C:\MyFile2.bat`.

6. In the **Run the post-build event** box, specify under what conditions to run the post-build event.

NOTE

To add lengthy syntax, or to select any build macros from the [Pre-build Event/Post-build Event Command Line Dialog Box](#), click the Ellipsis button (...) to display an edit box.

The build event syntax can include any command that is valid at a command prompt or in a .bat file. The name of a batch file should be preceded by `call` to ensure that all subsequent commands are executed.

Note If your pre-build or post-build event does not complete successfully, you can terminate the build by having your event action exit with a code other than zero (0), which indicates a successful action.

Example: How to Change Manifest Information by Using a Post-Build

Event

The following procedure shows how to set the minimum operating system version in the application manifest by using an .exe command that is called from a post-build event (the .exe.manifest file in the project directory). The minimum operating system version is a four-part number such as 4.10.0.0. To do this, the command will change the `<dependentOS>` section of the manifest:

```
<dependentOS>
  <osVersionInfo>
    <os majorVersion="4" minorVersion="10" buildNumber="0" servicePackMajor="0" />
  </osVersionInfo>
</dependentOS>
```

To create an .exe command to change the application manifest

1. Create a console application for the command. From the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, expand **Visual C#**, click **Windows**, and then click the **Console Application** template. Name the project `ChangeOSVersions.cs`.
3. In Program.cs, add the following line to the other `using` statements at the top of the file:

```
using System.Xml;
```

4. In the `ChangeOSVersions.cs` namespace, replace the `Program` class implementation with the following code:

```

class Program
{
    /// <summary>
    /// This function will set the minimum operating system version for a ClickOnce application.
    /// </summary>
    /// <param name="args">
    /// Command Line Arguments:
    /// 0 - Path to application manifest (.exe.manifest).
    /// 1 - Version of OS
    /// </param>
    static void Main(string[] args)
    {
        string applicationManifestPath = args[0];
        Console.WriteLine("Application Manifest Path: " + applicationManifestPath);

        // Get version name.
        Version osVersion = null;
        if (args.Length >= 2 ){
            osVersion = new Version(args[1]);
        }else{
            throw new ArgumentException("OS Version not specified.");
        }
        Console.WriteLine("Desired OS Version: " + osVersion.ToString());

        XmlDocument document;
        XmlNamespaceManager namespaceManager;
        namespaceManager = new XmlNamespaceManager(new NameTable());
        namespaceManager.AddNamespace("asmv1", "urn:schemas-microsoft-com:asm.v1");
        namespaceManager.AddNamespace("asmv2", "urn:schemas-microsoft-com:asm.v2");

        document = new XmlDocument();
        document.Load(applicationManifestPath);

        string baseXPath;
        baseXPath = "/asmv1:assembly/asmv2:dependency/asmv2:dependentOS/asmv2:osVersionInfo/asmv2:os";

        // Change minimum required operating system version.
        XmlNode node;
        node = document.SelectSingleNode(baseXPath, namespaceManager);
        node.Attributes["majorVersion"].Value = osVersion.Major.ToString();
        node.Attributes["minorVersion"].Value = osVersion.Minor.ToString();
        node.Attributes["buildNumber"].Value = osVersion.Build.ToString();
        node.Attributes["servicePackMajor"].Value = osVersion.Revision.ToString();

        document.Save(applicationManifestPath);
    }
}

```

The command takes two arguments: the path of the application manifest (that is, the folder in which the build process creates the manifest, typically Projectname.publish), and the new operating system version.

5. Build the project. On the **Build** menu, click **Build Solution**.

6. Copy the .exe file to a directory such as `C:\TEMP\ChangeOSVersionVB.exe`.

Next, invoke this command in a post-build event to modify the application manifest.

To invoke a post-build event to modify the application manifest

1. Create a Windows application for the project to be published. From the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, expand **Visual C#**, click **Windows Classic Desktop**, and then click the **Windows Forms App** template. Name the project `CSWinApp`.
3. With the project selected in **Solution Explorer**, on the **Project** menu, click **Properties**.

4. In the Project Designer, locate the **Publish** page and set **Publishing location** to `C:\TEMP\`.

5. Publish the project by clicking **Publish Now**.

The manifest file will be built and put in `C:\TEMP\CSWinApp_1_0_0\CSWinApp.exe.manifest`. To view the manifest, right-click the file, click **Open with**, select **Select the program from a list**, and then click **NotePad**.

Search in the file for the `<osVersionInfo>` element. For example, the version might be:

```
<os majorVersion="4" minorVersion="10" buildNumber="0" servicePackMajor="0" />
```

6. In the Project Designer, click the **Build Events** tab and click the **Edit Post-build** button.

7. In the **Post-build Event Command Line** box, type the following command:

```
C:\TEMP\ChangeOSVersionCS.exe "$(TargetPath).manifest" 5.1.2600.0
```

When you build the project, this command will change the minimum operating system version in the application manifest to 5.1.2600.0.

Because the `$(TargetPath)` macro expresses the full path for the executable being created, the `$(TargetPath).manifest` will specify the application manifest created in the bin directory. Publishing will copy this manifest to the publishing location that you set earlier.

8. Publish the project again. Go to the **Publish** page and click **Publish Now**.

View the manifest again. To view the manifest, open the publish directory, right-click the file, click **Open with**, select **Select the program from a list**, and then click **NotePad**.

The version should now read:

```
<os majorVersion="5" minorVersion="1" buildNumber="2600" servicePackMajor="0" />
```

See Also

[Build Events Page, Project Designer \(C#\)](#)

[Pre-build Event/Post-build Event Command Line Dialog Box](#)

[How to: Specify Build Events \(Visual Basic\)](#)

[Compiling and Building](#)

Configuring Warnings in Visual Basic

12/22/2017 • 5 min to read • [Edit Online](#)

The Visual Basic compiler includes a set of warnings about code that may cause run-time errors. You can use that information to write cleaner, faster, better code with fewer bugs. For example, the compiler will produce a warning when the user attempts to invoke a member of an unassigned object variable, return from a function without setting the return value, or execute a `Try` block with errors in the logic to catch exceptions.

Sometimes the compiler provides extra logic on the user's behalf so that the user can focus on the task at hand, rather than on anticipating possible errors. In previous versions of Visual Basic, `Option Strict` was used to limit the additional logic that the Visual Basic compiler provides. Configuring warnings allows you to limit this logic in a more granular way, at the level of the individual warnings.

You may want to customize your project and turn off some warnings not pertinent to your application while turning other warnings into errors. This page explains how to turn individual warnings on and off.

Turning Warnings Off and On

There are two different ways to configure warnings: you can configure them using the **Project Designer**, or you can use the `/warnaserror` and `/nowarn` compiler options.

The **Compile** tab of the **Project Designer** page allows you to turn warnings on and off. Select the **Disable All Warnings** check box to disable all warnings; select the **Treat All Warnings as Errors** to treat all warnings as errors. Some individual warnings can be toggled as error or warning as desired in the displayed table.

When **Option Strict** is set to **Off**, **Option Strict** related warnings cannot be treated independently of each other. When **Option Strict** is set to **On**, the associated warnings are treated as errors, no matter what their status is.

When **Option Strict** is set to **Custom** by specifying `/optionstrict:custom` in the command line compiler, **Option Strict** warnings can be toggled on or off independently.

The `/warnaserror` command-line option of the compiler can also be used to specify whether warnings are treated as errors. You can add a comma delimited list to this option to specify which warnings should be treated as errors or warnings by using + or -. The following table details the possible options.

| COMMAND-LINE OPTION | SPECIFIES |
|---|--|
| <code>/warnaserror+</code> | Treat all warnings as errors |
| <code>/warnaserror -</code> | Do not treat as warnings as errors. This is the default. |
| <code>/warnaserror+:<warning list></code> | Treat specific warnings as errors, listed by their error ID number in a comma delimited list. |
| <code>/warnaserror-:<warning list></code> | Do not treat specific warnings as errors, listed by their error ID number in a comma delimited list. |
| <code>/nowarn</code> | Do not report warnings. |
| <code>/nowarn:<warning list></code> | Do not report specified warnings, listed by their error ID number in a comma delimited list. |

The warning list contains the error ID numbers of the warnings that should be treated as errors, which can be used with the command-line options to turn specific warnings on or off. If the warning list contains an invalid number, an error is reported.

Examples

This table of examples of command line arguments describes what each argument does.

| ARGUMENT | DESCRIPTION |
|---|--|
| <code>vbc /warnaserror</code> | Specifies that all warnings should be treated as errors. |
| <code>vbc /warnaserror:42024</code> | Specifies that warning 42024 should be treated as an error. |
| <code>vbc /warnaserror:42024,42025</code> | Specifies that warnings 42024 and 42025 should be treated as errors. |
| <code>vbc /nowarn</code> | Specifies that no warnings should be reported. |
| <code>vbc /nowarn:42024</code> | Specifies that warning 42024 should not be reported. |
| <code>vbc /nowarn:42024,42025</code> | Specifies that warnings 42024 and 42025 should not be reported. |

Types of Warnings

Following is a list of warnings that you might want to treat as errors.

Implicit Conversion Warning

Generated for instances of implicit conversion. They do not include implicit conversions from an intrinsic numeric type to a string when using the `&` operator. Default for new projects is off.

ID: 42016

Late bound Method Invocation and Overload Resolution Warning

Generated for instances of late binding. Default for new projects is off.

ID: 42017

Operands of Type Object Warnings

Generated when operands of type `Object` occur that would create an error with `Option Strict On`. Default for new projects is on.

ID: 42018 and 42019

Declarations Require 'As' Clause Warnings

Generated when a variable, function, or property declaration lacking an `As` clause would have created an error with `Option Strict On`. Variables that do not have a type assigned to them are assumed to be type `Object`. Default for new projects is on.

ID: 42020 (variable declaration), 42021 (function declaration), and 42022 (property declaration).

Possible Null Reference Exception Warnings

Generated when a variable is used before it has been assigned a value. Default for new projects is on.

ID: 42104, 42030

Unused Local Variable Warning

Generated when a local variable is declared but never referred to. Default is on.

ID: 42024

Access of Shared member through Instance Variable Warning

Generated when accessing a shared member through an instance may have side effects, or when accessing a shared member through an instance variable is not the right-hand side of an expression or is being passed in as a parameter. Default for new projects is on.

ID: 42025

Recursive Operator or Property Access Warnings

Generated when the body of a routine uses the same operator or property it is defined in. Default for new projects is on.

ID: 42004 (operator), 42026 (property)

Function or Operator without Return Value Warning

Generated when the function or operator does not have a return value specified. This includes omitting a `Set` to the implicit local variable with the same name as the function. Default for new projects is on.

ID: 42105 (function), 42016 (operator)

Overloads Modifier Used in a Module Warning

Generated when `Overloads` is used in a `Module`. Default for new projects is on.

ID: 42028

Duplicate or Overlapping Catch Blocks Warnings

Generated when a `Catch` block is never reached due to its relation to other `catch` blocks that have been defined. Default for new projects is on.

ID: 42029, 42031

See Also

[Error Types](#)

[Try...Catch...Finally Statement](#)

[/nowarn](#)

[/warnaserror \(Visual Basic\)](#)

[Compile Page, Project Designer \(Visual Basic\)](#)

[Compiler Warnings That Are Off by Default](#)

How to: Disable the Hosting Process

1/26/2018 • 1 min to read • [Edit Online](#)

Calls to certain APIs can be affected when the hosting process is enabled. In these cases, it is necessary to disable the hosting process to return the correct results.

To disable the hosting process

1. Open an executable project in Visual Studio. Projects that do not produce executables (for example, class library or service projects) do not have this option.
2. On the **Project** menu, click **Properties**.
3. Click the **Debug** tab.
4. Clear the **Enable the Visual Studio hosting process** check box.

When the hosting process is disabled, several debugging features are unavailable or experience decreased performance. For more information, see [Debugging and the Hosting Process](#).

In general, when the hosting process is disabled:

- The time needed to begin debugging .NET Framework applications increases.
- Design-time expression evaluation is unavailable.
- Partial trust debugging is unavailable.

See also

[Debugging and the Hosting Process](#)

[Hosting Process \(vhost.exe\)](#)

Hosting Process (vhost.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

The hosting process is a feature in Visual Studio that improves debugging performance, enables partial trust debugging, and enables design time expression evaluation. The hosting process files contain vhost in the file name and are placed in the output folder of your project. For more information, see [Debugging and the Hosting Process](#).

NOTE

Hosting process files (.vhost.exe) are for use by Visual Studio and should not be run directly or deployed with your application.

Improved Debugging Performance

The hosting process creates an application domain and associates the debugger with the application. Performing these tasks can introduce a noticeable delay between the time debugging is started and the time the application begins running. The hosting process helps increase performance by creating the application domain and associating the debugger in the background, and saving the application domain and debugger state between runs of the application. For more information on application domains, see [Application Domains](#).

Partial Trust Debugging

An application can be specified as a partial trust application in the [Security page](#) of the **Project Designer**. Debugging a partial trust application requires special initialization of the application domain. This initialization is handled by the hosting process.

Design-Time Expression Evaluation

Design-time expression evaluation enables you to test code from the **Immediate** window without having to run the application. The hosting process executes this code during design time expression evaluation. For more information, see [Immediate Window](#).

See Also

[Debugging and the Hosting Process](#)

[How to: Disable the Hosting Process](#)

[Immediate Window](#)

[Application Domains](#)

Walkthrough: Creating a Multiple-Computer Build Environment

2/21/2018 • 12 min to read • [Edit Online](#)

You can create a build environment within your organization by installing Visual Studio on a host computer and then copying various files and settings to another computer so that it can participate in builds. You don't have to install Visual Studio on the other computer.

This document does not confer rights to redistribute the software externally or to provide build environments to third parties.

Disclaimer

This document is provided on a "as-is" basis. While we have tested the steps outlined, we are not able to exhaustively test every configuration. We will attempt to keep the document current with any additional information learned. Information and views expressed in this document, including URL and other Internet website references, may change without notice. Microsoft makes no warranties, express or implied, with respect to the information provided here. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

You have no obligation to give Microsoft any suggestions, comments or other feedback ("Feedback") relating to this document. However, any Feedback you voluntarily provide may be used in Microsoft Products and related specifications or other documentation (collectively, "Microsoft Offerings") which in turn may be relied upon by other third parties to develop their own products. Accordingly, if you do give Microsoft Feedback on any version of this document or the Microsoft Offerings to which they apply, you agree: (a) Microsoft may freely use, reproduce, license, distribute, and otherwise commercialize your Feedback in any Microsoft Offering; (b) You also grant third parties, without charge, only those patent rights necessary to enable other products to use or interface with any specific parts of a Microsoft Product that incorporate Your Feedback; and (c) You will not give Microsoft any Feedback (i) that you have reason to believe is subject to any patent, copyright or other intellectual property claim or right of any third party; or (ii) subject to license terms which seek to require any Microsoft Offering incorporating or derived from such Feedback, or other Microsoft intellectual property, to be licensed to or otherwise shared with any third party.

This walkthrough has been validated against the following operating systems, by executing MSBuild on the command line and by using Team Foundation Build.

- Windows 8 (x86 and x64)
- Windows 7 Ultimate
- Windows Server 2008 R2 Standard

After you complete the steps in this walkthrough, you can use the multiple-computer environment to build these kinds of apps:

- C++ desktop apps that use the Windows 8 SDK
- Visual Basic or C# desktop apps that target the .NET Framework 4.5

The multiple-computer environment can't be used to build these kinds of apps:

- UWP apps. To build UWP apps, you must install Visual Studio on the build computer.
- Desktop apps that target the .NET Framework 4 or earlier. To build these kinds of apps, you must install either Visual Studio or the .NET Reference Assemblies and Tools (from the Windows 7.1 SDK) on the build computer.

This walkthrough is divided into these parts:

- [Installing software on the computers](#)
- [Copying files from the host computer to the build computer](#)
- [Creating registry settings](#)
- [Setting environment variables on the build computer](#)
- [Installing MSBuild assemblies to the Global Assembly Cache \(GAC\) on the build computer](#)
- [Building projects](#)
- [Creating the build environment so that it can be checked into source control](#)

Prerequisites

- Visual Studio with the .NET desktop development workload installed.

Installing software on the computers

First, set up the host computer and then set up the build computer.

By installing Visual Studio on the host computer, you create the files and settings that you will copy to the build computer later. You can install Visual Studio on an x86 or an x64 computer, but the architecture of the build computer must match the architecture of the host computer.

1. On the host computer, install Visual Studio.
2. On the build computer, install the .NET Framework 4.5. To verify that it's installed, make sure that the value of the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full@Version starts with "4.5".

Copying files from the host computer to the build computer

This section covers the copying of specific files, compilers, build tools, MSBuild assets, and registry settings from the host computer to the build computer. These instructions assume that you've installed Visual Studio in the default location on the host computer; if you installed in another location, adjust the steps accordingly.

- On an x86 computer, the default location is C:\Program Files\Microsoft Visual Studio 11.0\
- On an x64 computer, the default location is C:\Program Files (x86)\Microsoft Visual Studio 11.0\

Notice that the name of the Program Files folder depends on the operating system that's installed. On an x86 computer, the name is \Program Files\; on an x64 computer, the name is \Program Files (x86)\. Irrespective of the system architecture, this walkthrough refers to the Program Files folder as %ProgramFiles%.

NOTE

On the build computer, all of the relevant files must be on the same drive; however, the drive letter for that drive can be different than the drive letter for the drive where Visual Studio is installed on the host computer. In any case, you must account for the location of files when you create registry entries as described later in this document.

To copy the Windows SDK files to the build computer

1. If you have only the Windows SDK for Windows 8 installed, copy these folders recursively from the host computer to the build computer:

- %ProgramFiles%\Windows Kits\8.0\bin\
- %ProgramFiles%\Windows Kits\8.0\Catalogs\
- %ProgramFiles%\Windows Kits\8.0\DesignTime\
- %ProgramFiles%\Windows Kits\8.0\include\
- %ProgramFiles%\Windows Kits\8.0\Lib\
- %ProgramFiles%\Windows Kits\8.0\Redist\
- %ProgramFiles%\Windows Kits\8.0\References\

If you also have these other Windows 8 kits...

- Microsoft Windows Assessment and Deployment Kit
- Microsoft Windows Driver Kit
- Microsoft Windows Hardware Certification Kit

...they might have installed files into the %ProgramFiles%\Windows Kits\8.0\ folders that are listed in the previous step, and their license terms might not allow build-server rights for those files. Check the license terms for every installed Windows kit to verify whether files may be copied to your build computer. If the license terms don't allow build-server rights, then remove the files from the build computer.

2. Copy the following folders recursively from the host computer to the build computer:

- %ProgramFiles%\Microsoft SDKs\Windows\v8.0A\bin\NETFX 4.0 Tools\
- %ProgramFiles%\Common Files\Merge Modules\
- %ProgramFiles%\Microsoft Visual Studio 11.0\VC\
- %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\Tools\ProjectComponents\
- %ProgramFiles%\MSBuild\Microsoft.Cpp\v4.0\V110\
- %ProgramFiles%\Reference Assemblies\Microsoft\Framework\.NETCore\v4.5\
- %ProgramFiles%\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\

3. Copy these files from the host computer to the build computer:

- %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\IDE\msobj110.dll
- %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\IDE\mspdb110.dll
- %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\IDE\mspdbccore.dll
- %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\IDE\mspdbsrv.exe
- %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\IDE\msvcdis110.dll
- %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\Tools\makehm.exe
- %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\Tools\VCVarsQueryRegistry.bat
- %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\Tools\vsvars32.bat

4. The following Visual C++ runtime libraries are required only if you run build outputs on the build computer—for example, as part of automated testing. The files are typically located in subfolders under the %ProgramFiles%\Microsoft Visual Studio 11.0\VC\redist\x86\ or %ProgramFiles%\Microsoft Visual Studio 11.0\VC\redist\x64\ folder, depending on the system architecture. On x86 systems, copy the x86 binaries to the \Windows\System32\ folder. On x64 systems, copy the x86 binaries to the Windows\SysWOW64\ folder, and the x64 binaries to the Windows\System32\ folder.

- \Microsoft.VC110.ATL\atl110.dll
- \Microsoft.VC110.CRT\msvcp110.dll
- \Microsoft.VC110.CRT\msvcr110.dll
- \Microsoft.VC110.CXXAMP\vcamp110.dll
- \Microsoft.VC110.MFC\mfc110.dll
- \Microsoft.VC110.MFC\mfc110u.dll
- \Microsoft.VC110.MFC\mfcm110.dll
- \Microsoft.VC110.MFC\mfcm110u.dll
- \Microsoft.VC110.MFCLOC\mfc110chs.dll
- \Microsoft.VC110.MFCLOC\mfc110cht.dll
- \Microsoft.VC110.MFCLOC\mfc110deu.dll
- \Microsoft.VC110.MFCLOC\mfc110enu.dll
- \Microsoft.VC110.MFCLOC\mfc110esn.dll
- \Microsoft.VC110.MFCLOC\mfc110fra.dll
- \Microsoft.VC110.MFCLOC\mfc110ita.dll
- \Microsoft.VC110.MFCLOC\mfc110jpn.dll
- \Microsoft.VC110.MFCLOC\mfc110kor.dll
- \Microsoft.VC110.MFCLOC\mfc110rus.dll
- \Microsoft.VC110.OPENMP\vcomp110.dll

5. Copy only the following files from the \Debug_NonRedist\x86\ or \Debug_NonRedist\x64\ folder to the build computer, as described in [Preparing a Test Machine To Run a Debug Executable](#). No other files may be copied.

- \Microsoft.VC110.DebugCRT\msvcp110d.dll
- \Microsoft.VC110.DebugCRT\msvcr110d.dll
- \Microsoft.VC110.DebugCXXAMP\vcamp110d.dll
- \Microsoft.VC110.DebugMFC\mfc110d.dll
- \Microsoft.VC110.DebugMFC\mfc110ud.dll
- \Microsoft.VC110.DebugMFC\mfcm110d.dll
- \Microsoft.VC110.DebugMFC\mfcm110ud.dll
- \Microsoft.VC110.DebugOpenMP\vcomp110d.dll

Creating registry settings

You must create registry entries to configure settings for MSBuild.

To create registry settings

1. Identify the parent folder for registry entries. All of the registry entries are created under the same parent key. On an x86 computer, the parent key is HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\. On an x64 computer the parent key is HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\. Irrespective of the system architecture, this walkthrough refers to the parent key as %RegistryRoot%.

NOTE

If the architecture of your host computer differs from that of your build computer, make sure to use the appropriate parent key on each computer. This is especially important if you're automating the export process.

Also, if you're using a different drive letter on the build computer than the one that you're using on the host computer, make sure to change the values of the registry entries to match.

2. Create the following registry entries on the build computer. All of these entries are strings (Type == "REG_SZ" in the registry). Set the values of these entries the same as the values of the comparable entries on the host computer.

- %RegistryRoot%\NETFramework\v4.0.30319\AssemblyFoldersEx\VCMSBuild Public Assemblies@(Default)
- %RegistryRoot%\Microsoft SDKs\Windows\v8.0@InstallationFolder
- %RegistryRoot%\Microsoft SDKs\Windows\v8.0A@InstallationFolder
- %RegistryRoot%\Microsoft SDKs\Windows\v8.0A\WinSDK-NetFx40Tools@InstallationFolder
- %RegistryRoot%\Microsoft SDKs\Windows\v8.0A\WinSDK-NetFx40Tools-x86@InstallationFolder
- %RegistryRoot%\VisualStudio\11.0@Source Directories
- %RegistryRoot%\VisualStudio\11.0\Setup\VC@ProductDir
- %RegistryRoot%\VisualStudio\SxS\VC7@FrameworkDir32
- %RegistryRoot%\VisualStudio\SxS\VC7@FrameworkDir64
- %RegistryRoot%\VisualStudio\SxS\VC7@FrameworkVer32
- %RegistryRoot%\VisualStudio\SxS\VC7@FrameworkVer64
- %RegistryRoot%\VisualStudio\SxS\VC7@11.0
- %RegistryRoot%\VisualStudio\SxS\VS7@11.0
- %RegistryRoot%\Windows Kits\Installed Roots@KitsRoot
- %RegistryRoot%\MSBuild\ToolsVersions\4.0\11.0@VCTargetsPath
- %RegistryRoot%\MSBuild\ToolsVersions\4.0\11.0@VCTargetsPath10
- %RegistryRoot%\MSBuild\ToolsVersions\4.0\11.0@VCTargetsPath11

On an x64 build computer, also create the following registry entry and refer to the host computer to determine how to set it.

- %RegistryRoot%\Microsoft SDKs\Windows\v8.0A\WinSDK-NetFx40Tools-x64@InstallationFolder

If your build computer is x64 and you want to use the 64-bit version of MSBuild, or if you're using Team Foundation Server Build Service on an x64 computer, you must create the following registry entries in the native 64-bit registry. Refer to the host computer to determine how to set these entries.

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\11.0\Setup\VS@ProductDir
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSBuild\ToolsVersions\4.0\11.0@VCTargetsPath
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSBuild\ToolsVersions\4.0\11.0@VCTargetsPath10
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSBuild\ToolsVersions\4.0\11.0@VCTargetsPath11

Setting environment variables on the build computer

To use MSBuild on the build computer, you must set the PATH environment variables. You can use vcvarsall.bat to set the variables, or you can manually configure them.

To use vcvarsall.bat to set environment variables

- Open a Command Prompt window on the build computer and run %Program Files%\Microsoft Visual Studio 11.0\VC\vcvarsall.bat. You can use a command-line argument to specify the toolset you want to use —x86, native x64, or x64 cross-compiler. If you don't specify a command-line argument, the x86 toolset is used.

This table describes the supported arguments for vcvarsall.bat:

| VCVARSALL.BAT ARGUMENT | COMPILER | BUILD COMPUTER ARCHITECTURE | BUILD OUTPUT ARCHITECTURE |
|------------------------|---------------|-----------------------------|---------------------------|
| x86 (default) | 32-bit Native | x86, x64 | x86 |
| x86_amd64 | x64 Cross | x86, x64 | x64 |
| amd64 | x64 Native | x64 | x64 |

If vcvarsall.bat runs successfully—that is, no error message is displayed—you can skip the next step and continue at the [Installing MSBuild assemblies to the Global Assembly Cache \(GAC\) on the build computer](#) section of this document.

To manually set environment variables

1. To manually configure the command-line environment, add this path to the PATH environment variable:
 - %Program Files%\Microsoft Visual Studio 11.0\Common7\IDE
2. Optionally, you can also add the following paths to the PATH variable to make it easier to use MSBuild to build your solutions.

If you want to use the native 32-bit MSBuild, add these paths to the PATH variable:

- %Program Files%\Microsoft SDKs\Windows\v8.0A\bin\NETFX 4.0 Tools
- %windir%\Microsoft.NET\Framework\v4.0.30319

If you want to use the native 64-bit MSBuild, add these paths to the PATH variable:

- %Program Files%\Microsoft SDKs\Windows\v8.0A\bin\NETFX 4.0 Tools\x64
- %windir%\Microsoft.NET\Framework64\v4.0.30319

Installing MSBuild assemblies to the Global Assembly Cache (GAC) on

the build computer

MSBuild requires some additional assemblies to be installed to the GAC on the build computer.

1. Copy the following assemblies from the host computer to the build computer. Because they will be installed to the GAC, it doesn't matter where you put them on the build computer.
 - %ProgramFiles%\MSBuild\Microsoft.Cpp\v4.0\v110\Microsoft.Build.CPPTasks.Common.v110.dll
 - %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\IDE\CommonExtensions\Microsoft\VC\Project\Microsoft.VisualStudio.Project.VCProjectEngine.dll
 - %ProgramFiles%\Microsoft Visual Studio 11.0\Common7\IDE\PublicAssemblies\Microsoft.VisualStudio.VCProjectEngine.dll
2. To install the assemblies to the GAC, locate gacutil.exe on the build computer—typically, it's in %ProgramFiles%\Microsoft SDKs\Windows\v8.0A\bin\NETFX 4.0 Tools\. If you can't locate this folder, repeat the steps in the [Copying files from the host computer to the build computer](#) section of this walkthrough.

Open a Command Prompt window that has administrative rights and run this command for each file:

gacutil -i <file>

NOTE

A reboot may be required for an assembly to fully install into the GAC.

Building projects

You can use Team Foundation Build to build Visual Studio 2012 projects and solutions, or you can build them on the command line. When you use Team Foundation Build to build projects, it invokes the MSBuild executable that corresponds to the system architecture. On the command line, you can use either 32-bit MSBuild or 64-bit MSBuild, and you can choose the architecture of MSBuild by setting the PATH environment variable or by directly invoking the architecture-specific MSBuild executable.

To use msbuild.exe at the command prompt, run the following command, in which *solution.sln* is a placeholder for the name of your solution.

msbuild solution.sln

For more information about how to use MSBuild on the command line, see [Command-Line Reference](#).

NOTE

To build Visual Studio 2012 projects, you must use the "v110" Platform Toolset. If you don't want to edit the Visual Studio 2012 project files, you can set the Platform Toolset by using this command-line argument:

msbuild solution.sln /p:PlatformToolset=v110

Creating the build environment so that it can be checked into source control

You can create a build environment that can be deployed to various computers and doesn't require GAC'ing files or modifying registry settings. The following steps are just one way to accomplish this. Adapt these steps to the unique characteristics of your build environment.

NOTE

You must disable incremental building so that tracker.exe will not throw an error during a build. To disable incremental building, set this build parameter:

```
msbuild solution.sln /p:Track FileAccess=false
```

1. Create a "Depot" directory on the host computer.

These steps refer to the directory as %Depot%.

2. Copy the directories and files as described in the [Copying files from the host computer to the build computer](#) section of this walkthrough, except paste them under the %Depot% directory that you just created. For example, copy from %ProgramFiles%\Windows Kits\8.0\bin\ to %Depot%\Windows Kits\8.0\bin\.

3. When the files are pasted in %Depot%, make these changes:

- In %Depot%\MSBuild\Microsoft.Cpp\v4.0\v110\Microsoft.CPP.Targets, \Microsoft.Cpp.InvalidPlatforms.targets\, \Microsoft.cppbuild.targets\, and \Microsoft.CppCommon.targets\, change every instance of

```
AssemblyName="Microsoft.Build.CppTasks.Common.v110, Version=4.0.0.0, Culture=neutral,  
PublicKeyToken=b03f5f7f11d50a3a"
```

to

```
AssemblyFile="$(VCTargetsPath11)Microsoft.Build.CppTasks.Common.v110.dll".
```

The former naming relies on the assembly being GAC'ed.

- In %Depot% \MSBuild\Microsoft.Cpp\v4.0\v110\Microsoft.CPPClean.Targets, change every instance of

```
AssemblyName="Microsoft.Build.CppTasks.Common.v110, Version=4.0.0.0, Culture=neutral,  
PublicKeyToken=b03f5f7f11d50a3a"
```

to

```
AssemblyFile="$(VCTargetsPath11)Microsoft.Build.CppTasks.Common.v110.dll".
```

4. Create a .props file—for example, Partner.AutoImports.props—and put it at the root of the folder that contains your projects. This file is used to set variables that are used by MSBuild to find various resources. If the variables are not set by this file, they are set by other .props files and .targets files that rely on registry values. Because we aren't setting any registry values, these variables would be empty and the build would fail. Instead, add this to Partner.AutoImports.props:

```

<?xml version="1.0" encoding="utf-8"?>
<!-- This file must be imported by all project files at the top of the project file. -->
<Project ToolsVersion="4.0"
  xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
<PropertyGroup>
  <VCTargetsPath>$(DepotRoot)MSBuild\Microsoft.Cpp\v4.0\v110\</VCTargetsPath>
  <VCTargetsPath11>$(DepotRoot)MSBuild\Microsoft.Cpp\v4.0\v110\</VCTargetsPath11>
  <MSBuildExtensionsPath>$(DepotRoot)MSBuild</MSBuildExtensionsPath>
  <MSBuildExtensionsPath32>$(DepotRoot)MSBuild</MSBuildExtensionsPath32>
  <VCInstallDir_110>$(DepotRoot)Microsoft Visual Studio 11.0\VC\</VCInstallDir_110>
  <VCInstallDir>$(VCInstallDir_110)\</VCInstallDir>
  <WindowsKitRoot>$(DepotRoot)Windows Kits\</WindowsKitRoot>
  <WindowsSDK80Path>$(WindowsKitRoot)\</WindowsSDK80Path>
  <WindowsSdkDir_80>$(WindowsKitRoot)8.0\</WindowsSdkDir_80>
  <WindowsSdkDir>$(WindowsSDKDir_80)\</WindowsSdkDir>
  <WindowsSdkDir_80A>$(DepotRoot)Microsoft SDKs\Windows\v8.0A\</WindowsSdkDir_80A>
</PropertyGroup>
</Project>

```

5. In each of your project files, add the following line at the top, after the `<Project Default Targets...>` line.

```

<Import Project="$(MSBuild]::GetDirectoryNameOfFileAbove($(MSBuildThisFileDirectory),
  Partner.AutoImports.props))\Partner.AutoImports.props"/>

```

6. Change the command-line environment as follows:

- Set Depot=*location of the Depot directory that you created in step 1*
- Set path=%path%;*location of MSBuild on the computer*;%Depot%\Windows\System32;%Depot%\Windows\SysWOW64;%Depot%\Microsoft Visual Studio 11.0\Common7\IDE\

For native 64-bit building, point to the 64-bit MSBuild.

See also

[Preparing a Test Machine To Run a Debug Executable](#)

[Command-Line Reference](#)

Feature Tour of the Visual Studio Debugger

1/4/2018 • 10 min to read • [Edit Online](#)

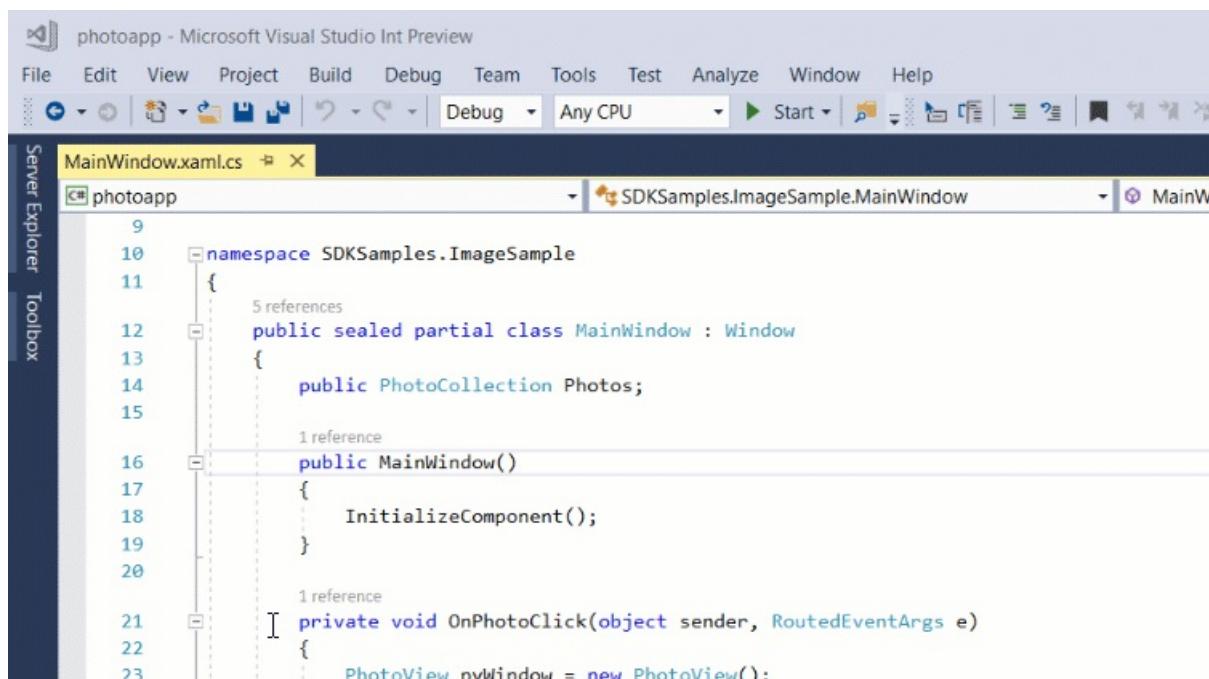
This topic introduces the features of the Visual Studio debugger. If you want to follow along by opening your own app in Visual Studio, you can do that, or you can follow along with a sample app using the [Beginner's Guide](#).

The features described here are applicable to C#, C++, Visual Basic, JavaScript, and other languages supported by Visual Studio (except where noted).

Set a breakpoint and start the debugger

To debug, you need to start your app with the debugger attached to the app process. F5 (**Debug > Start Debugging**) is the most common way to do that. However, right now you may not have set any breakpoints to examine your app code, so we will do that first and then start debugging.

If you have a file open in the code editor, you can set a breakpoint by clicking in the margin to the left of a line of code.



Press F5 (**Debug > Start Debugging**) and the debugger runs to the first breakpoint that it encounters. If the app is not yet running, F5 starts the debugger and stops at the first breakpoint.

Breakpoints are a useful feature when you know the line of code or the section of code that you want to examine in detail.

Navigate code in the debugger using step commands

We provide the keyboard shortcuts for most commands because they make navigation of your app code quicker. (Equivalent commands such as menu commands are shown in parentheses.)

To start your app with the debugger attached, press F11 (**Debug > Step Into**). F11 is the **Step Into** command and advances the app execution one statement at a time. When you start the app with F11, the debugger breaks on the first statement that gets executed.

```
12     public sealed partial class MainWindow : Window
13     {
14         public PhotoCollection Photos;
15
16         public MainWindow()
17         {
18             InitializeComponent(); // Yellow arrow points here
19         }
20     }
```

The yellow arrow represents the statement on which the debugger paused, which also suspends app execution at the same point (this statement has not yet executed).

F11 is a good way to examine the execution flow in the most detail. (To move faster through code, we show you some other options as well.) By default, the debugger skips over non-user code (if you want more details, see [Just My Code](#)).

NOTE

In managed code, you will see a dialog box asking if you want to be notified when you automatically step over properties and operators (default behavior). If you want to change the setting later, disable **Step over properties and operators** setting in the **Tools > Options** menu under **Debugging**.

Step over code to skip functions

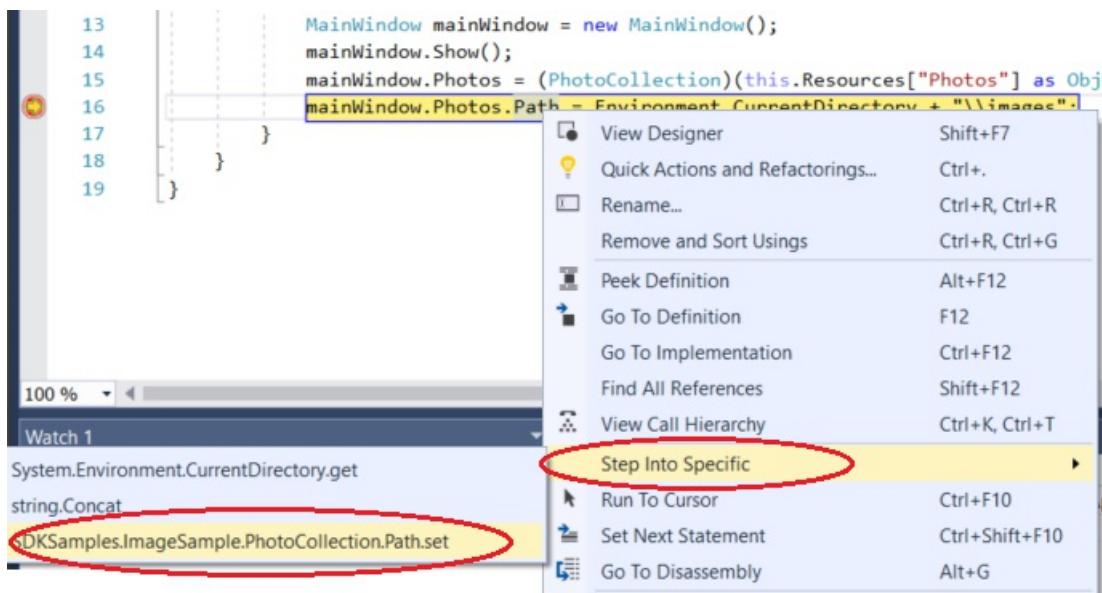
When you are on a line of code that is a function or method call, you can press F10 (**Debug > Step Over**) instead of F11.

F10 advances the debugger without stepping into functions or methods in your app code (the code still executes). By pressing F10, you can skip over code that you're not interested in. This way, you can quickly get to code that you are more interested in.

Step into a property

As mentioned earlier, by default the debugger skips over managed properties and fields, but the **Step Into Specific** command allows you to override this behavior.

Right-click on a property or field and choose **Step Into Specific**, then choose one of the available options.



In this example, **Step Into Specific** gets us to the code for `Path.set`.

```
55 public string Path
56 {
57     set
58     {
59         _directory = new DirectoryInfo(value);
60         Update();
61     }
62     get { return _directory.FullName; }
63 }
```

Run to a point in your code quickly using the mouse

While in the debugger, hover over a line of code until the **Run to Click** (Run execution to here) button appears on the left.

```
55 public string Path
56 {
57     set
58     {
59         _directory = new DirectoryInfo(value);
60          Update();
61     }
62     get { return _directory.FullName; }
63 }
```

NOTE

The **Run to Click** (Run execution to here) button is new in Visual Studio 2017.

Click the **Run to Click** (Run execution to here) button. The debugger advances to the line of code where you clicked.

Using this button is similar to setting a temporary breakpoint. This command is also handy for getting around quickly within a visible region of app code. You can use **Run to Click** in any open file.

Advance the debugger out of the current function

Sometimes, you might want to continue your debugging session but advance the debugger all the way through the current function.

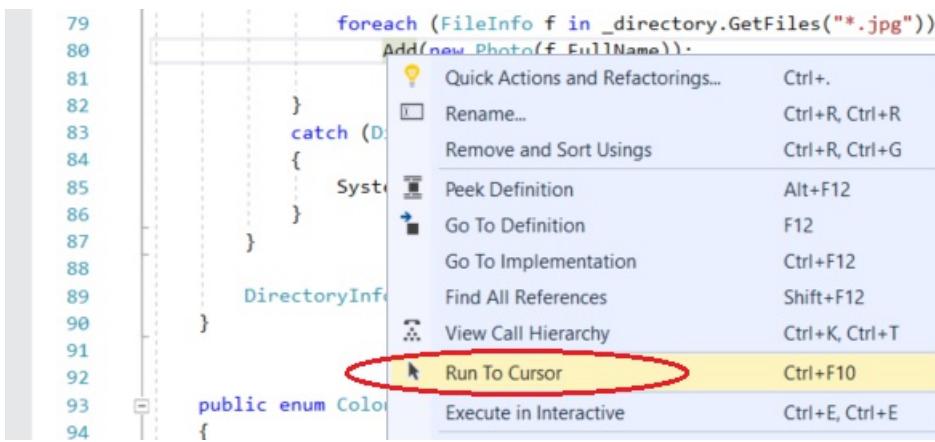
Press Shift + F11 (or **Debug > Step Out**).

This command resumes app execution (and advances the debugger) until the current function returns.

Run to cursor

Stop the debugger by pressing the **Stop Debugging** red button or Shift + F5.

Right-click a line of code in your app and choose **Run to Cursor**. This command starts debugging and sets a temporary breakpoint on the current line of code.



If you have set breakpoints, the debugger pauses on the first breakpoint that it hits.

Press F5 until you reach the line of code where you selected **Run to Cursor**.

This command is useful when you are editing code and want to quickly set a temporary breakpoint and start the debugger.

NOTE

You can use **Run to Cursor** in the **Call Stack** window while you are debugging.

Restart your app quickly

Click the **Restart**  button in the Debug Toolbar (**Ctrl + Shift + F5**).

When you press **Restart**, it saves time versus stopping the app and restarting the debugger. The debugger pauses at the first breakpoint that is hit by executing code.

If you do want to stop the debugger and get back into the code editor, you can press the red stop  button instead of **Restart**.

Inspect variables with data tips

Now that you know your way around a little, you have a good opportunity to start inspecting your app state (variables) with the debugger. Features that allow you to inspect variables are some of the most useful features of the debugger, and there are different ways to do it. Often, when you try to debug an issue, you are attempting to find out whether variables are storing the values that you expect them to have in a particular app state.

While paused in the debugger, hover over an object with the mouse and you see its default property value (in this example, the file name `market_031.jpg` is the default property value).

```

74     private void Update()
75     {
76         this.Clear();
77         try
78         {
79             foreach (FileInfo f in _directory.GetFiles("*.jpg"))
80                 Add(new Photo(f.FullName)); ≤ 3ms elapsed
81         }
82         catch (DirectoryNotFoundException)
83         {
84             System.Windows.MessageBox.Show("No Such Directory");
85         }
86     }
87 }
88

```

Expand the object to see all its properties (such as the `FullPath` property in this example).

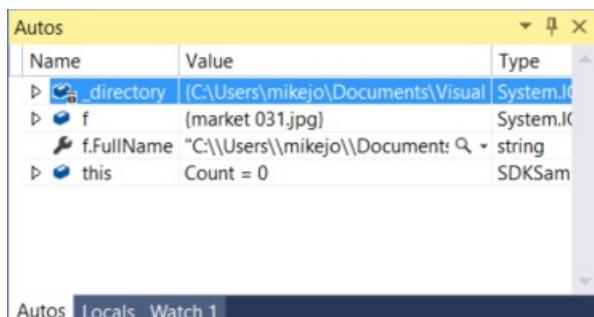
Often, when debugging, you want a quick way to check property values on objects, and the data tips are a good way to do it.

TIP

In most supported languages, you can edit code in the middle of a debugging session. For more info, see [Edit and Continue](#).

Inspect variables with the Autos and Locals windows

While debugging, look at the **Autos** window at the bottom of the code editor.

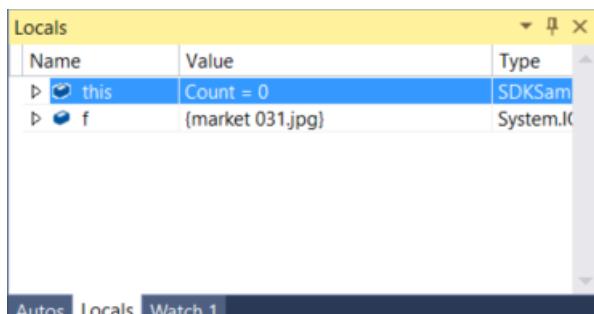


In the **Autos** window, you see variables along with their current value and their type. The **Autos** window shows all variables used on the current line or the preceding line (In C++, the window shows variables in the preceding three lines of code. Check documentation for language-specific behavior).

NOTE

In JavaScript, the **Locals** window is supported but not the **Autos** window.

Next, look at the **Locals** window. The **Locals** window shows you the variables that are currently in scope.



In this example, the `this` object and the object `f` are in scope. For more info, see [Inspect Variables in the Autos and Locals Windows](#).

Set a watch

You can use a **Watch** window to specify a variable (or an expression) that you want to keep an eye on.

While debugging, right-click an object and choose **Add Watch**.

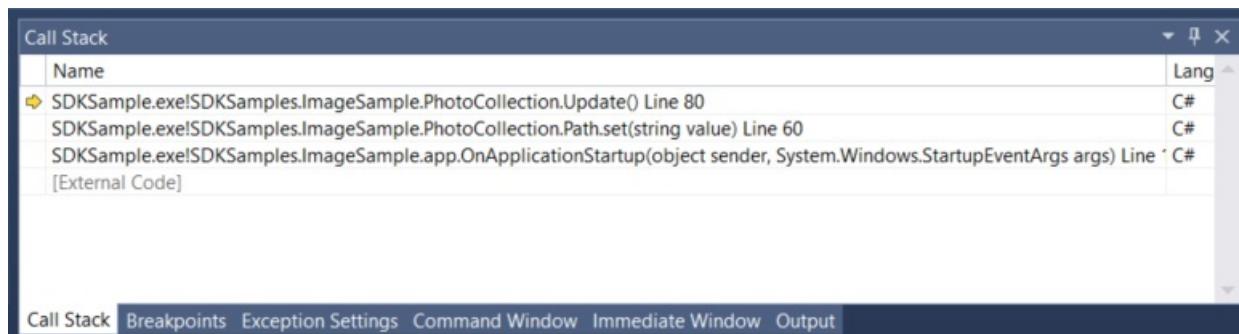


In this example, you have a watch set on the `File` object, and you can see its value change as you move through the debugger. Unlike the other variable windows, the **Watch** windows always show the variables that you are watching (they're grayed out when out of scope).

For more info, see [Set a Watch using the Watch and QuickWatch Windows](#)

Examine the call stack

Click the **Call Stack** window while you are debugging, which is by default open in the lower right pane.



The **Call Stack** window shows the order in which methods and functions are getting called. The top line shows the current function (the `Update` method in this example). The second line shows that `Update` was called from the `Path.set` property, and so on. The call stack is a good way to examine and understand the execution flow of an app.

NOTE

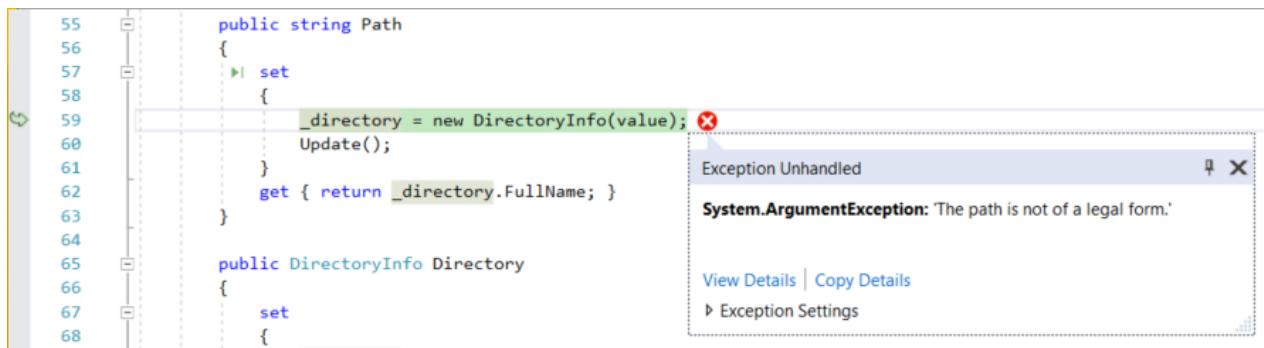
The **Call Stack** window is similar to the Debug perspective in some IDEs like Eclipse.

You can double-click a line of code to go look at that source code and that also changes the current scope being inspected by the debugger. This does not advance the debugger.

You can also use right-click menus from the **Call Stack** window to do other things. For example, you can insert breakpoints into specific functions, restart your app using **Run to Cursor**, and to go examine source code. See [How to: Examine the Call Stack](#).

Examine an exception

When your app throws an exception, the debugger takes you to the line of code that threw the exception.



In this example, the **Exception Helper** shows you a `System.Argument` exception and an error message that says that the path is not a legal form. So, we know the error occurred on a method or function argument.

In this example, the `DirectoryInfo` call gave the error on the empty string stored in the `value` variable.

The Exception Helper is a great feature that can help you debug errors. You can also do things like view error details and add a watch from the Exception Helper. Or, if needed, you can change conditions for throwing the particular exception.

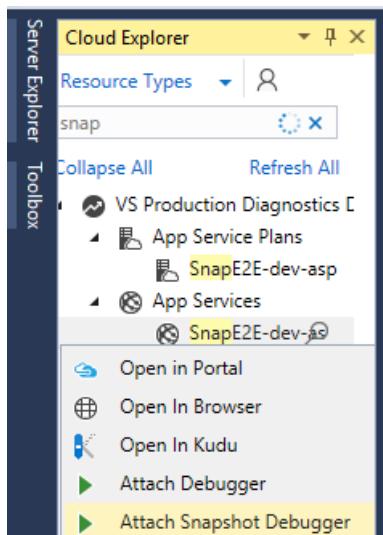
NOTE

The Exception Helper replaces the Exception Assistant in Visual Studio 2017.

Expand the **Exception Settings** node to see more options on how to handle this exception type, but you don't need to change anything for this tour!

Debug live ASP.NET apps in Azure App Service

the **Snapshot Debugger** takes a snapshot of your in-production apps when code that you are interested in executes. To instruct the debugger to take a snapshot, you set snappoints and logpoints in your code. The debugger lets you see exactly what went wrong, without impacting traffic of your production application. The Snapshot Debugger can help you dramatically reduce the time it takes to resolve issues that occur in production environments.



Snapshot collection is available for ASP.NET applications running in Azure App Service. ASP.NET applications must be running on .NET Framework 4.6.1 or later, and ASP.NET Core applications must be running on .NET Core 2.0 or

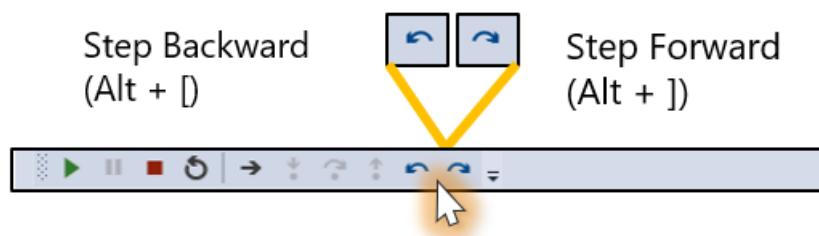
later on Windows.

For more information, see [Debug live ASP.NET apps using the Snapshot Debugger](#).

View snapshots with IntelliTrace step-back (Visual Studio Enterprise)

IntelliTrace step-back automatically takes a snapshot of your application at every breakpoint and debugger step event. The recorded snapshots enable you to go back to previous breakpoints or steps and view the state of the application as it was in the past. IntelliTrace step-back can save you time when you want to see the previous application state but don't want to restart debugging or recreate the desired app state.

You can navigate and view snapshots by using the **Step Backward** and **Step Forward** buttons in the Debug toolbar. These buttons navigate the events that appear in the **Events** tab in the **Diagnostic Tools** window.



For more information, see the [View snapshots using IntelliTrace step-back](#) page.

More features to look at

- [Debugger Tips and Tricks](#) Learn how to increase your productivity with the debugger.
- [Edit and Continue](#) For a subset of languages (C#, C++, Visual Basic), the Edit and Continue feature allows you to edit code in the middle of a debugging session.
- [Debug Multithreaded Applications](#) Describes how to debug multithreaded applications.
- [Remote Debugging](#) Describes how to debug apps running on other machines or devices.
- [IntelliTrace](#) Describes the IntelliTrace feature in Visual Studio Enterprise. You can use it to record and trace your code's execution history.
- [Network Usage](#) Describes a profiling tool that you can use to debug web services and other network resources in Universal Windows Apps (UWP). Use the tool to examine payloads.
- [Debug Interface Access SDK](#) Describes the Microsoft Debug Interface Access Software Development Kit (DIA SDK). The DIA SDK provides access to debug information stored in program database (.pdb) files generated by Microsoft postcompiler tools.

See Also

[Debugging in Visual Studio](#)

Testing tools in Visual Studio

3/20/2018 • 1 min to read • [Edit Online](#)

Visual Studio testing tools can help you and your team develop and sustain high standards of code excellence.

- The **Test Explorer** window makes it easy to integrate [unit tests](#) into your development practice. You can use the Microsoft unit test framework or one of several third-party and open source frameworks.
- [IntelliTest](#) automatically generates unit tests and test data for your managed code.
- [Code coverage](#) determines what proportion of your project's code is actually being tested by coded tests such as unit tests.
- [Microsoft Fakes](#) help you isolate the code you are testing by replacing other parts of the application with stubs or shims.
- [Live Unit Testing](#) automatically runs unit tests in the background, and graphically displays code coverage and test results in the Visual Studio code editor.
- [Coded UI tests](#) enable you to test your application through its user interface.
- [Load testing](#) simulates load on a server application by running unit tests and Web performance tests.

NOTE

Unit testing is available in all editions of Visual Studio. Other testing tools, such as live unit testing, IntelliTest, and Coded UI tests are only available in Visual Studio Enterprise edition. For more information about editions see [Compare Visual Studio 2017 IDEs](#).

Related scenarios

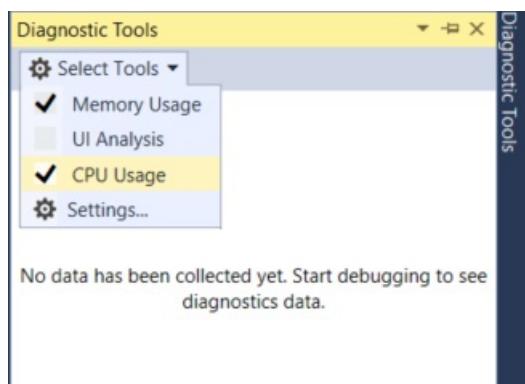
- [Exploratory & manual testing \(VSTS\)](#)
- [Load testing \(VSTS\)](#)
- [Continuous testing \(VSTS\)](#)
- [DevOps overview for Team Services and TFS \(VSTS\)](#)
- [Code analysis tools](#)

Profiling Feature Tour

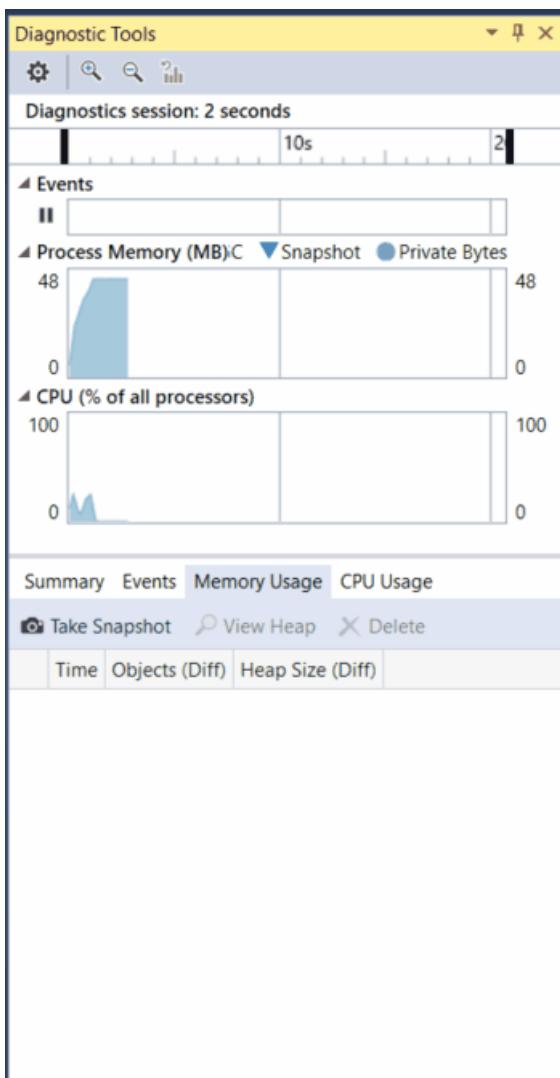
3/12/2018 • 8 min to read • [Edit Online](#)

Visual Studio provides a variety of profiling tools to help you diagnose different kinds of performance issues depending on your app type.

The profiling tools that you can access during a debugging session are available in the Diagnostic Tools window. The Diagnostic Tools window appears automatically unless you have turned it off. To bring up the window, click **Debug / Windows / Show Diagnostic Tools**. With the window open, you can select tools for which you want to collect data.



While you are debugging, you can use the **Diagnostic Tools** window to analyze CPU and memory usage, and you can view events that show performance-related information.

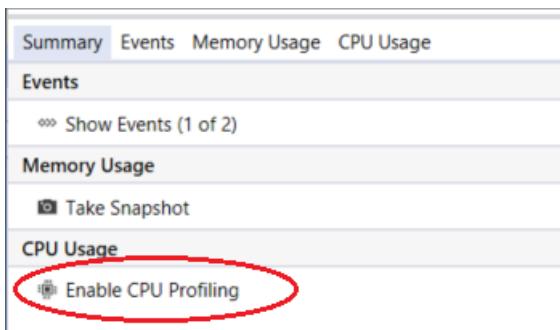


The **Diagnostic Tools** window is often the preferred way to profile apps, but for Release builds you can also do a post-mortem analysis of your app instead. If you want more information on different approaches, see [Running Profiling Tools With or Without the Debugger](#). To see profiling tool support for different app types, see [Which tool should I use?](#).

Analyze CPU Usage

The CPU Usage tool is a good place to start analyzing your app's performance. It will tell you more about CPU resources that your app is consuming. For a more detailed walkthrough of the CPU Usage tool, see [Beginner's Guide to Performance Profiling](#).

From the **Summary** view of the Diagnostic Tools, choose **Enable CPU Profiling** (you must be in a debugging session).



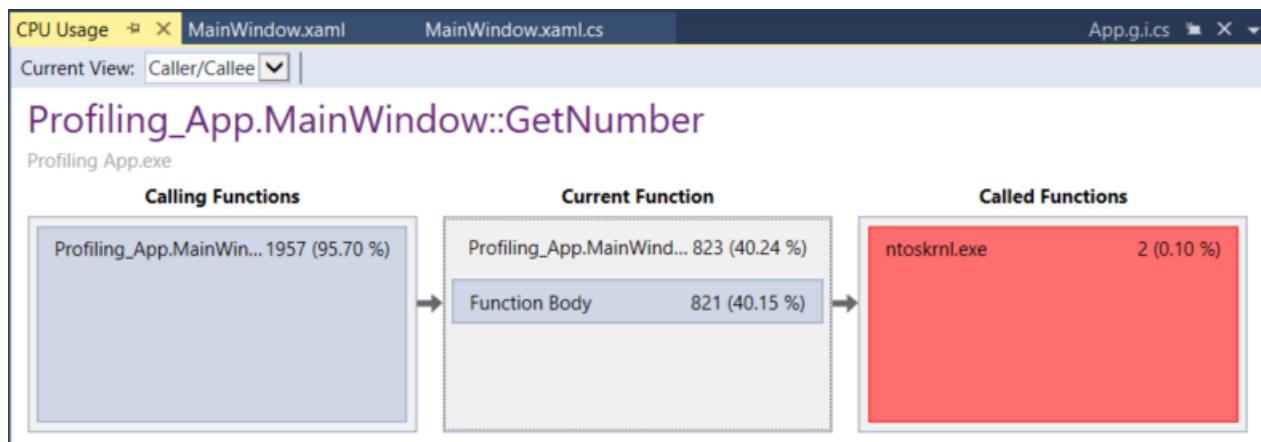
To use the tool most effectively, set two breakpoints in your code, one at the beginning and one at the end of the function or the region of code you want to analyze. Examine the profiling data when you are paused at the second

breakpoint.

The **CPU Usage** view shows you a list of functions ordered by longest running, with the longest running function at the top. This can help guide you to functions where performance bottlenecks are happening.

| Function Name | Total CPU [ms, %] |
|---|-------------------|
| Profiling_App.exe (PID: 14232) | 2369 (100.00 %) |
| Profiling_App.MainWindow::GetMaxNumberButton_Click | 2308 (97.43 %) |
| [External Call] System.Windows.Application.Run()\$##6000... | 2308 (97.43 %) |
| Profiling_App.App::Main | 2308 (97.43 %) |
| [External Call] System.Random.Next()\$##6000FF7 | 1255 (52.98 %) |
| Profiling_App.MainWindow::GetNumber | 1010 (42.63 %) |
| [External Call] System.Random.InternalSample()\$##6000FF6 | 38 (1.60 %) |
| [External Call] System.Windows.Controls.TextBox.set_Text(...) | 3 (0.13 %) |

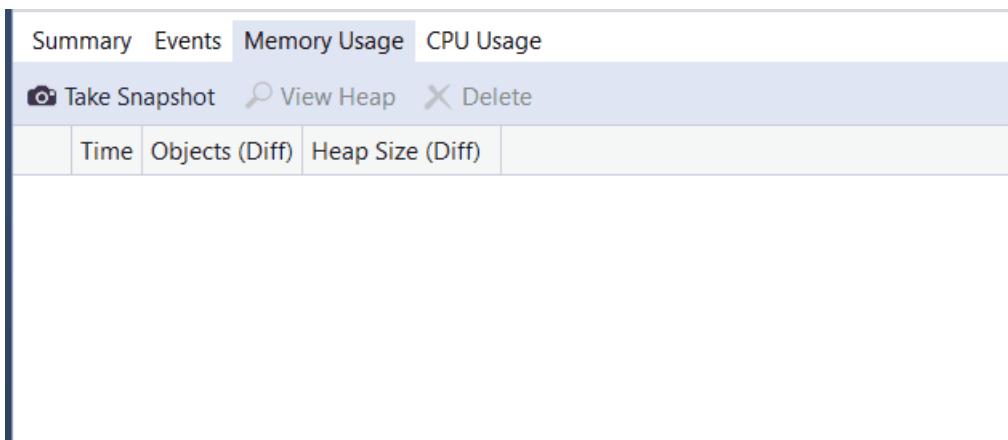
Double-click on a function that you are interested in, and you will see a more detailed three-pane "butterfly" view, with the selected function in the middle of the window, the calling function on the left, and called functions on the right. The **Function Body** section shows the total amount of time (and the percentage of time) spent in the function body excluding time spent in calling and called functions. This data can help you evaluate whether the function itself is a performance bottleneck.



Analyze Memory Usage

The Diagnostic Tools window also allows you to evaluate memory usage in your app. For example, you can look at the number and size of objects on the heap. For more detailed instructions to analyze memory, see [Analyze Memory Usage](#).

To analyze memory usage, you need to take at least one memory snapshot while you are debugging. Often, the best way to analyze memory is to take two snapshots; the first right before a suspected memory issue, and the second snapshot right after a suspected memory issue occurs. Then you can view a diff of the two snapshots and see exactly what changed.



When you select one of the arrow links, you are given a differential view of the heap (a red up arrow ↑ shows an increasing object count (left) or an increasing heap size (right)). If you click the right link, you get a differential heap view ordered by objects that increased the most in heap size. This can help you pinpoint memory problems. For example, in the illustration below, the bytes used by `ClassHandlersStore` objects increased by 3,492 bytes in the second snapshot.

Snapshot #2 Heap P...g App.exe (15.80s) ▸ × MainWindow.xaml MainWindow.xaml.cs

Managed Memory (Profiling App.exe) Compare to: Snapshot #1 ▼ ▴ Search

| Object Type | Count Diff. | Size Diff. (Bytes) | Inclusive Size Diff. (Bytes) | Count | Size (Bytes) | Inclusive Size (Bytes) |
|--|-------------|--------------------|------------------------------|-------|--------------|------------------------|
| <code>ClassHandlersStore</code> ⟳ | 0 | +3,492 | +3,428 | 32 | 9,296 | 23,580 |
| EventRoute | +3 | +1,368 | +1,524 | 4 | 1,808 | 2,016 |
| StylusPointPropertyInfo | +30 | +1,320 | +1,320 | 30 | 1,320 | 1,320 |
| StylusTouchDevice | +5 | +520 | +520 | 5 | 520 | 520 |
| DispatcherOperation | +5 | +420 | +1,780 | 8 | 672 | 2,528 |
| List<Int32> | +1 | +292 | +292 | 1 | 292 | 292 |
| TextTreeTextBlock | +1 | +244 | +244 | 1 | 244 | 244 |
| ExecutionContext | +5 | +220 | +320 | 12 | 528 | 744 |
| Task<Object> | +5 | +220 | +280 | 8 | 352 | 448 |
| Hashtable | +2 | +216 | +1,184 | 79 | 64,264 | 193,276 |
| PriorityItem<DispatcherOpe... | +5 | +160 | +1,152 | 8 | 256 | 1,272 |

Paths to Root | Referenced Types

| Object Type | Reference Count Diff. | Reference Count |
|---|-----------------------|-----------------|
| ◀ <code>ClassHandlersStore</code> | 0 | 32 |
| MS.Utility.DTypeMap [Static variable GlobalEventManager._dTypedClassListen... | | |

If you click the link on the left instead in the **Memory Usage** view, the heap view is organized by object count; the objects of a particular type that increased the most in number are shown at the top (sorted by **Count Diff** column).

Examine Performance Events

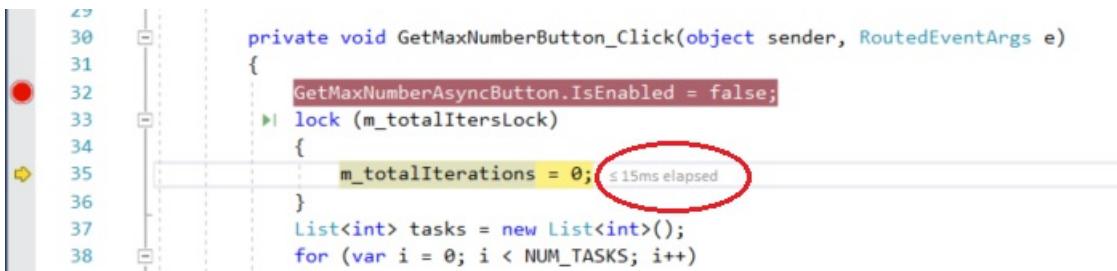
The **Events** view in the Diagnostic Tools shows you different events that occur while you are debugging, such as the setting of a breakpoint or a code stepping operation. You can check information such as the duration of the event (measured from when the debugger was last paused, or when the app started). For example, if you step through code (F10, F11), the **Events** view shows you the app runtime duration from the previous step operation to the current step.

| Event | Time | Duration | Thread |
|-----------------------------------|-------|--------------|--------|
| Breakpoint: Breakpoint Hit | 3.00s | 15ms [23784] | |
| Step: Step Recorded | 3.01s | 9ms [23784] | |
| Step: Step Recorded | 3.01s | 4ms [23784] | |
| Step: Step Recorded | 3.01s | 1ms [23784] | |
| Step: Step Recorded | 3.06s | 46ms [23784] | |
| Go to Source Code | | | |

NOTE

If you have Visual Studio Enterprise, you can also see [IntelliTrace events](#) in this tab.

The same events also show up in the code editor, which you can view as PerfTips.



A screenshot of a code editor showing a C# file with line numbers 29 through 38. A red circle marks a breakpoint at line 32. A yellow diamond marks a step point at line 30. A red oval highlights the line 35, which contains the assignment `m_totalIterations = 0;`. A tooltip to the right of the line shows "≤ 15ms elapsed".

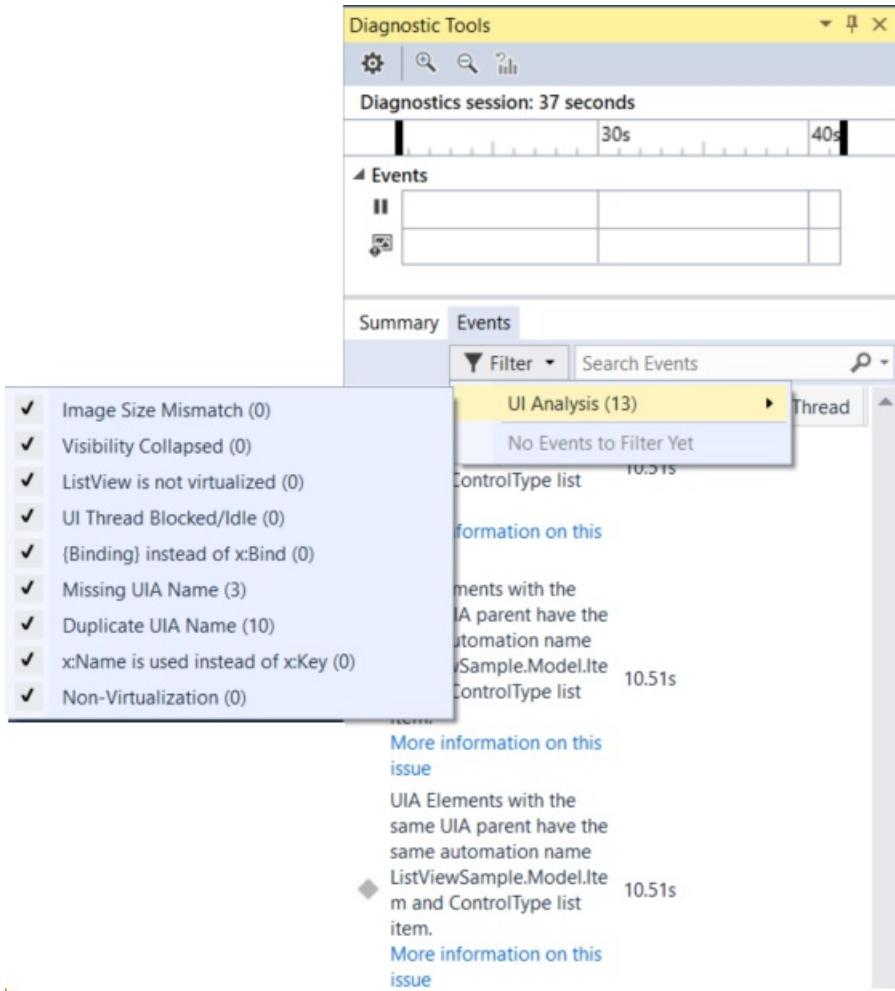
```

29
30
31
32
33
34
35 m_totalIterations = 0; ≤ 15ms elapsed
36
37
38

```

Examine UI Performance and Accessibility Events (UWP)

In your UWP apps, you can enable **UI Analysis** in the Diagnostic Tools window. The tool searches for common performance or accessibility issues and displays them in the **Events** view while you are debugging. The event descriptions provide information that can help resolve issues.



Profile Release Builds without the Debugger

Profiling tools like CPU Usage and Memory Usage can be used with the debugger (see earlier sections), or you can run profiling tools using the Performance Profiler, which is intended to provide analysis for **Release** builds. In the Performance Profiler, you can collect diagnostic info while the app is running, and then examine the collected information after the app is stopped. For more information on these different approaches, see [Running Profiling Tools With or Without the Debugger](#).

Report20170210-1117.diagsession MainWindow.xaml MainWindow.xaml.cs App.g.i.cs

> Analysis Target

Change Target ▾

Startup Project
Profiling App

⚠ Solution configuration is set to Debug. Switch to a Release configuration for more accurate results.

Available Tools [Show all tools](#)

Application Timeline
Examine where time is spent in your application. Useful when troubleshooting issues like low frame rate

CPU Usage
See where the CPU is spending time executing your code.
Useful when the CPU is the performance bottleneck

GPU Usage
Examine GPU usage in your DirectX application. Useful to determine whether the CPU or GPU is the performance bottleneck

Memory Usage
Investigate application memory to find issues such as memory leaks

Performance Wizard
CPU Sampling, Instrumentation, .NET Memory allocation, and Resource Contention

Start

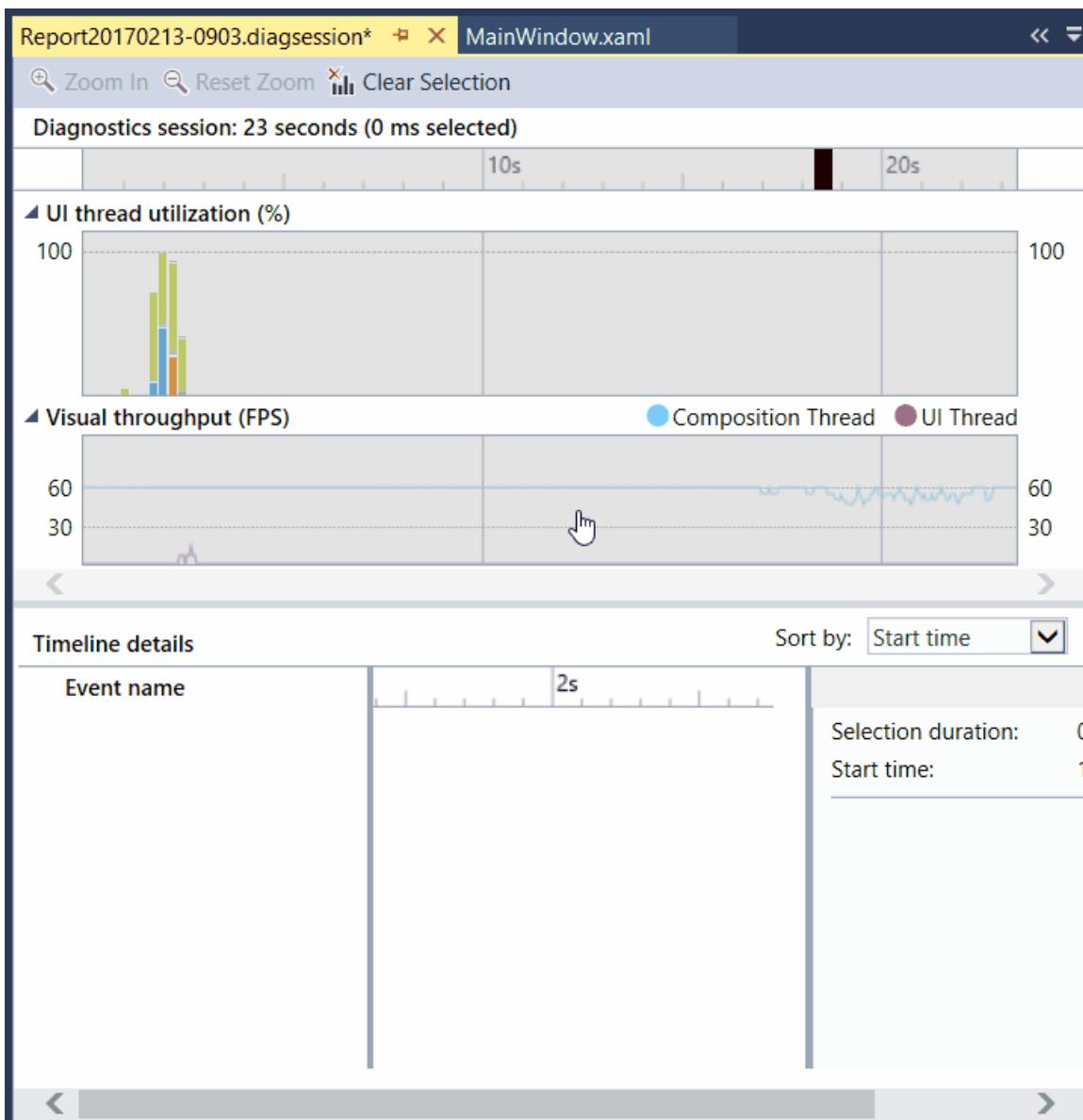
Open the Performance Profiler by choosing **Debug / Performance Profiler**.

The window will allow you to select multiple profiling tools in some scenarios. Tools such as CPU Usage may provide complementary data that you can use to help in your analysis.

Analyze Resource Consumption (XAML)

In XAML apps, such as Windows desktop WPF apps and UWP apps, you can analyze resource consumption using the Application Timeline tool. For example, you can analyze the time spent by your application preparing UI frames (layout and render), servicing network and disk requests, and in scenarios like application startup, page load, and Window resize. To use the tool, choose **Application Timeline** in the Performance Profiler, and then choose **Start**. In your app, go through the scenario with a suspected resource consumption issue, and then choose **Stop collection** to generate the report.

Low framerates in the **Visual throughput** graph may correspond to visual problems that you see when running your app. Similarly, high numbers in the **UI thread utilization** graph may also correspond to UI responsiveness issues. In the report, you can select a time period with a suspected performance issue, and then examine the detailed UI thread activities in the Timeline details view (lower pane).



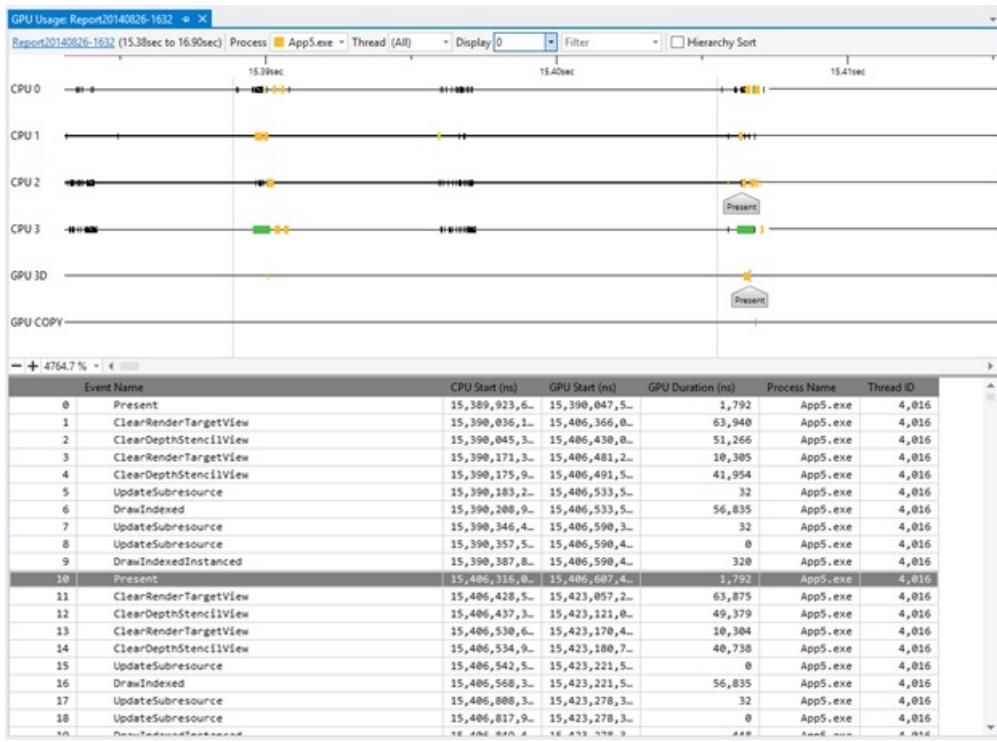
In the Timeline details view, you can find information such as the type of activity (or the UI element involved) along with the duration of the activity. For example, in the illustration, a **Layout** event for a Grid control takes 57.53 ms.

For more information, see [Application Timeline](#).

Analyze GPU Usage (Direct3D)

In Direct3D apps (Direct3D components must be in C++), you can examine activity on the GPU and analyze performance issues. For more information, see [GPU Usage](#). To use the tool, choose **GPU Usage** in the Performance Profiler, and then choose **Start**. In your app, go through the scenario that you're interested in profiling, and then choose **Stop collection** to generate a report.

When you select a time period in the graphs and choose **view details**, a detailed view appears in the lower pane. In the detailed view, you can examine how much activity is happening on each CPU and GPU. Select events in the lowest pane to get popups in the timeline. For example, select the **Present** event to view **Present** call popups. (The light gray vertical Vsync lines can be used as a reference to understand whether certain **Present** calls missed Vsync. There must be one **Present** call between every two Vsyncs in order for the app to steadily hit 60 FPS.)



You can also use the graphs to determine whether there are CPU bound or GPU bound performance bottlenecks.

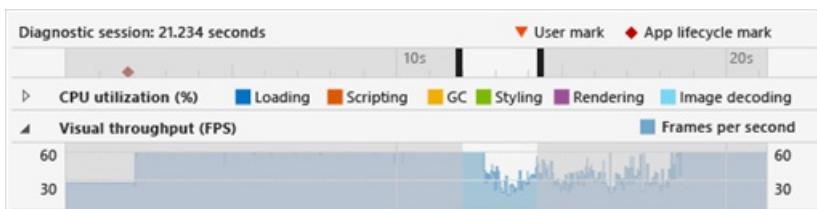
Analyze Performance (JavaScript)

For UWP apps, you can use the JavaScript Memory tool and the HTML UI Responsiveness tool.

The JavaScript Memory tool is similar to the Memory Usage tool available for other app types. You can use this tool to understand memory usage and find memory leaks in your app. For more details about the tool, see [JavaScript Memory](#).

| Snapshot #1 (20160307.diagsession) | | | | |
|------------------------------------|-------|------------|-------------------|-------|
| Types | Roots | Dominators | Identifier filter | |
| Identifier(s) | Type | Size | Retained size | Count |
| Function | | 30.23 KB | 26.71 KB | 460 |
| String | | 3.46 KB | 2.12 KB | 110 |
| HTMLDivElement | | 36.37 KB | | 56 |
| Array | | 2 KB | 2.01 KB | 24 |
| { cancel } | | 1.27 KB | | 18 |
| { _value } | | 1,008 B | 144 B | 14 |
| { handle, key, data } | | 1.12 KB | 3.18 KB | 13 |
| HTMLImageElement | | 249.19 KB | | 12 |
| Anonymous function | | 944 B | 2.23 KB | 9 |

To diagnose UI responsiveness, slow loading time, and slow visual updates in UWP apps, use the HTML UI Responsiveness tool. Usage is similar to the Application Timeline tool for other app types. For more information, see [HTML UI responsiveness](#).



Analyze Network Usage (UWP)

In UWP apps, you can analyze network operations performed using the `Windows.Web.Http` API. This tool may help you to resolve issues like access and authentication problems, incorrect cache-use, and poor display and download performance. To use the tool, choose **Network** in the Performance Profiler, and then choose **Start**. In your app, go through the scenario that uses `Windows.Web.Http`, and then choose **Stop collection** to generate the report.



Select an operation in the summary view to view more details.

| Headers | Body | Parameters | Cookies | Timings |
|---|------|------------|---------|---------|
| Request URL: http://localhost:46789/WCFSERVICE.svc/Get... | | | | |
| Request Method: POST | | | | |
| Status Code: 200 / OK | | | | |
| Request Headers | | | | |
| Accept: application/json | | | | |
| Connection: Keep-Alive | | | | |
| Content-Length: 27 | | | | |
| Content-Type: application/json; charset=utf-8 | | | | |
| Host: localhost:46789 | | | | |
| Response Headers | | | | |
| Cache-Control: private | | | | |
| Content-Length: 55 | | | | |
| Content-Type: application/json; charset=utf-8 | | | | |
| Date: Tue, 21 Feb 2017 19:01:50 GMT | | | | |
| Server: Microsoft-IIS/10.0 | | | | |
| X-AspNet-Version: 4.0.30319 | | | | |
| X-Powered-By: ASP.NET | | | | |
| X-SourceFiles: =?UTF-8?B?QzpcVXNlcnNcbWlrZWpvXER... | | | | |

For more information, see [Network Usage](#).

Analyze Performance (Legacy Tools)

If you need features such as instrumentation that are not currently present in CPU Usage or Memory Usage tools, and you are running desktop or ASP.NET apps, you can use the Performance Explorer for profiling. (Not supported in UWP apps). For more info, see [Performance Explorer](#).

Hot Path

| Function Name | Inclusive Samples % | Exclusive Samples % |
|---|---------------------|---------------------|
| Profiling_App.App.Main | 99.56 | 0.00 |
| System.Windows.Application.Run | 99.56 | 7.74 |
| Profiling_App.MainWindow.GetMaxNumberButton_Click | 91.77 | 0.00 |
| System.Random.Next | 65.09 | 65.09 |
| Profiling_App.MainWindow.GetNumber | 22.83 | 22.83 |

Related Views: [Call Tree](#) [Functions](#)

Which Tool Should I Use?

Here is a table that lists the different tools Visual Studio offers and the different project types you can use them with:

| PERFORMANCE TOOL | WINDOWS DESKTOP | UWP | ASP.NET/ASP.NET CORE |
|------------------------|---|---|---|
| Memory Usage | yes | yes | yes |
| CPU Usage | yes (see note) | yes | yes (see note) |
| GPU Usage | yes | yes | no |
| Application Timeline | yes | yes | no |
| PerfTips | yes | yes for XAML, no for HTML | yes |
| Performance Explorer | yes | no | yes |
| IntelliTrace | .NET with Visual Studio Enterprise only | .NET with Visual Studio Enterprise only | .NET with Visual Studio Enterprise only |
| Network Usage | no | yes | no |
| HTML UI responsiveness | no | yes for HTML, no for XAML | no |
| JavaScript Memory | no | yes for HTML, no for XAML | no |

NOTE

For .NET Core and ASP.NET Core, the CPU Usage tool currently does not provide accurate results with portable PBDs. Use full PDBs instead.

See Also

[Debugging in Visual Studio](#)

4 min to read •

Deployment Overview in Visual Studio

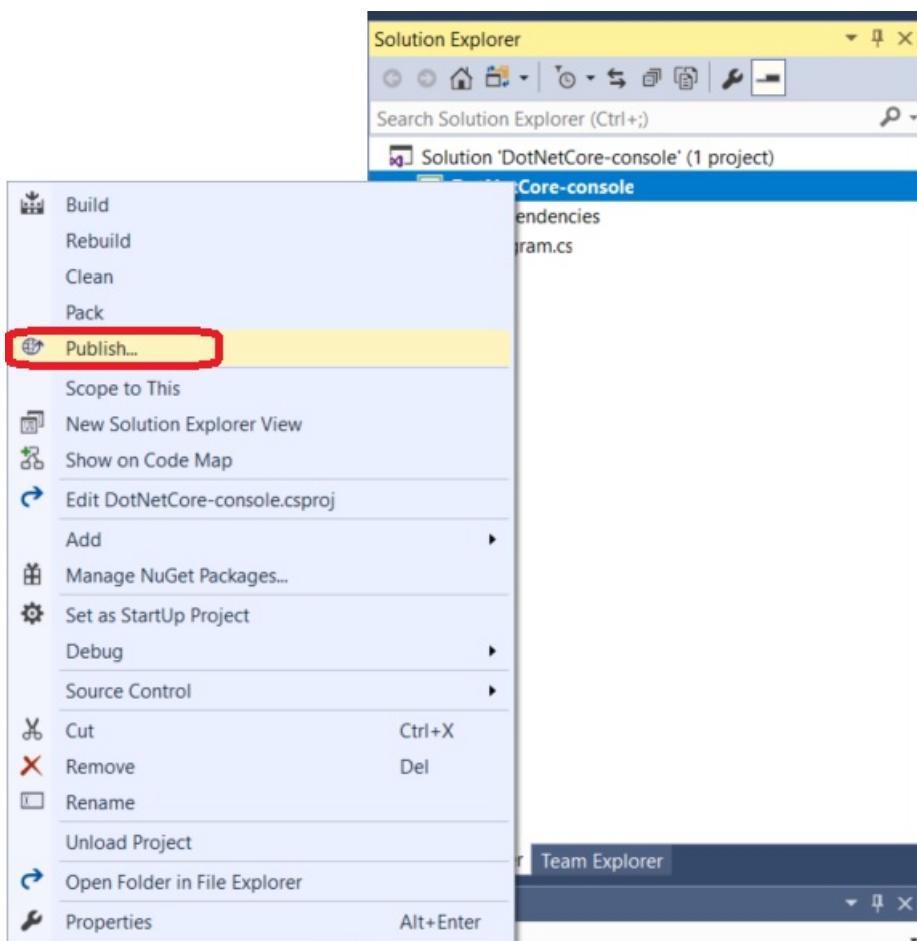
1/10/2018 • 3 min to read • [Edit Online](#)

By deploying an application, service, or component, you distribute it for installation on other computers, devices, servers, or in the cloud. You choose the appropriate method in Visual Studio for the type of deployment that you need. (Many app types support other deployment tools such as command line deployment or NuGet that are not described here.)

See the Tutorials for step-by-step instructions.

Deploy to local folder

- **ASP.NET, ASP.NET Core, Node.js, Python, and .NET Core:** Use the Publish tool to deploy to a local folder. The exact options available depend on your app type. In Solution Explorer, right-click your project and choose **Publish**, and then choose **Folder**. For more information, see [Deploy to a local folder](#).

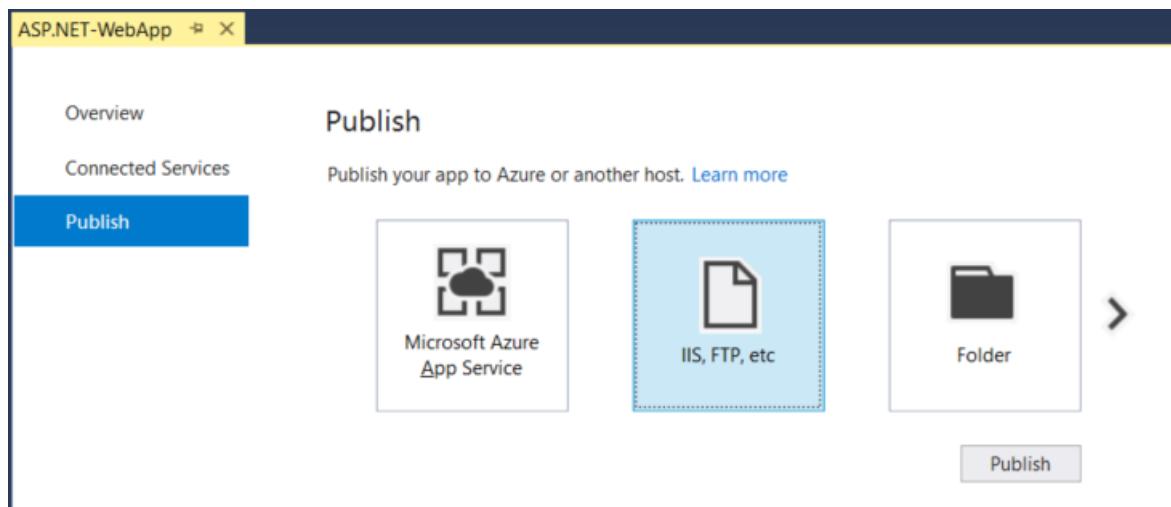


- **Visual C++ runtime:** You can deploy the Visual C++ runtime using local deployment or static linking. For more information, see [Deploying Native Desktop Applications \(Visual C++\)](#).

Publish to Web or deploy to network share

- **ASP.NET, ASP.NET Core, Node.js, Python, and .NET Core:** You can use the Publish tool to deploy to a website using FTP or Web Deploy. For more information, see [Deploy to a web site](#).

In Solution Explorer, right-click the project and choose **Publish**. In the Publish tool, choose the option you want and follow the configuration steps.

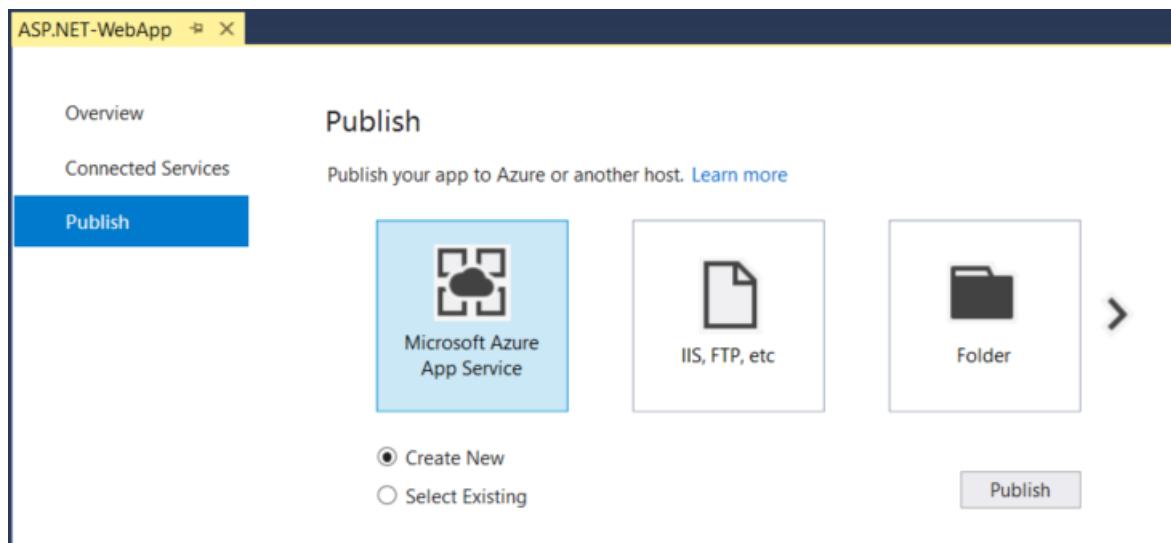


You can also deploy ASP.NET applications and services in a number of other ways. For more information, see [Deploying ASP.NET web applications and services](#).

- **Visual C++ runtime:** You can deploy the Visual C++ runtime using central deployment. For more information, see [Deploying Native Desktop Applications \(Visual C++\)](#).
- **Windows desktop** You can publish a Windows desktop application to a web server or a network file share using ClickOnce deployment. Users can then install the application with a single click. For more information, see [Deploy a desktop app using ClickOnce](#) and [Deploy a native app using ClickOnce](#).

Publish to Azure

- **ASP.NET, ASP.NET Core, Python, Node.js, and .NET Core** web applications: You can use the Publish tool to quickly deploy apps to Azure App Service or to an Azure Virtual Machine. In Solution Explorer, right-click the project and choose **Publish**. In the Publish dialog box, choose either **Microsoft Azure App Service** or **Microsoft Azure Virtual Machines**, and then follow the configuration steps.



To publish to an Azure Virtual Machine, scroll right and select **Microsoft Azure Virtual Machines**.

For a quick introduction, see [Publish to Azure](#). Also, see [Publish an ASP.NET Core app to Azure](#). For deployment using Git, see [Continuous deployment of ASP.NET Core to Azure with Git](#).

NOTE

If you do not already have an Azure account, you can [sign up here](#).

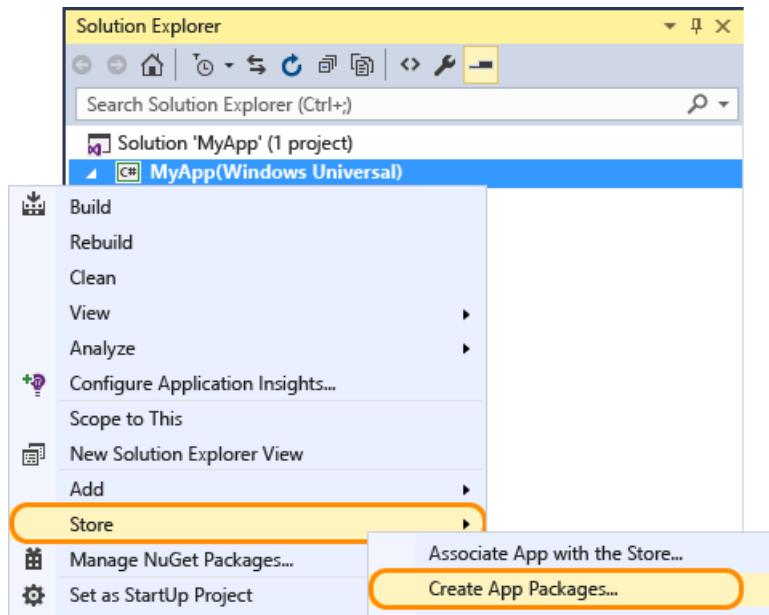
- Other **Azure services**: See the specific [Azure service](#) documentation for different deployment options that

may be supported by Visual Studio.

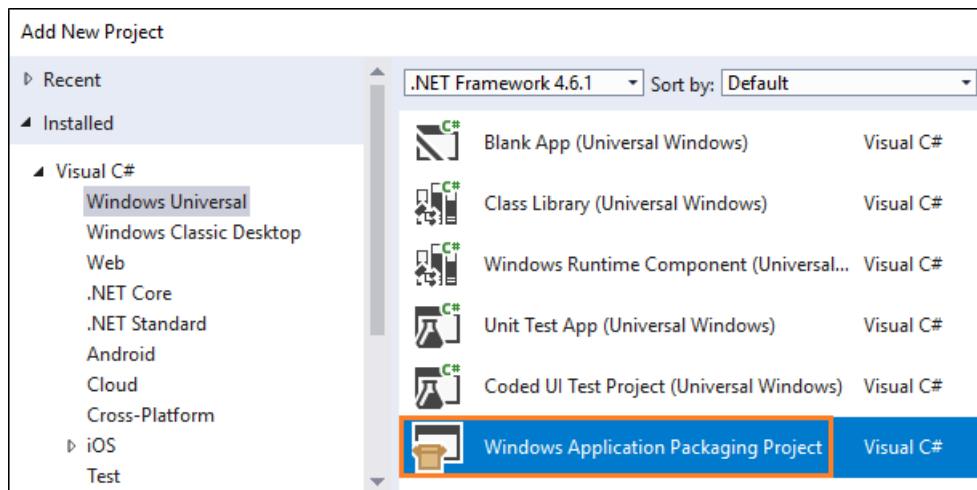
Publish to Microsoft Store

From Visual Studio, you can create app packages for deployment to Microsoft Store.

- **UWP:** You can package your app and deploy it using menu items. For more information, see [Package a UWP app by using Visual Studio](#).



- **Windows desktop:** You can deploy to the Microsoft Store using the Desktop Bridge starting in Visual Studio 2017 version 15.4. To do this, start by creating a Windows Application Packaging Project. For more information, see [Package a desktop app for Microsoft Store \(Desktop Bridge\)](#).



Create an Installer package (Windows client)

- An MSI-based WiX installer can be created using the [WiX Toolset Visual Studio 2017 Extension](#).
- [InstallShield](#) from Flexera Software may be used with Visual Studio 2017 (Community Edition not supported). Note that InstallShield Limited Edition is no longer included with Visual Studio and is not supported in Visual Studio 2017; check with [Flexera Software](#) about future availability.
- If you want to create a Setup project (vdproj), install the [Visual Studio 2017 Installer Projects extension](#).
- You can install prerequisite components for desktop applications by configuring a generic installer, which is known as a bootstrapper. For more information, see [Application Deployment Prerequisites](#).

Deploy to test lab

You can enable more sophisticated development and testing by deploying your applications into virtual

environments. For more information, see [Test on a lab environment](#).

DevOps deployment

In a team environment, you can use Visual Studio Team Services (VSTS) to enable continuous deployment of your app. For more information, see [Build and Release](#) and [Deploy to Azure](#).

Deployment for other app types

| APP TYPE | DEPLOYMENT SCENARIO | LINK |
|-----------------------------|--|--|
| Office app | You can publish an add-in for Office from Visual Studio. | Deploy and publish your Office add-in |
| WCF or OData service | Other applications can use WCF RIA services that you deploy to a web server. | Developing and deploying WCF Data Services |
| LightSwitch | LightSwitch is no longer supported in Visual Studio 2017, but can still be deployed from Visual Studio 2015 and earlier. | Deploying LightSwitch Applications |

Visual Studio SDK

12/22/2017 • 2 min to read • [Edit Online](#)

The Visual Studio SDK helps you extend Visual Studio features or integrate new features into Visual Studio. You can distribute your extensions to other users, as well as to the Visual Studio Marketplace. The following are some of the ways in which you can extend Visual Studio:

- Add commands, buttons, menus, and other UI elements to the IDE
- Add tool windows for new functionality
- Extend IntelliSense for a given language, or provide IntelliSense for new programming languages
- Use light bulbs to provide hints and suggestions that help developers write better code
- Enable support for a new language
- Add a custom project type
- Reach millions of developers via the Visual Studio Marketplace

If you've never written a Visual Studio extension before, you should find more information about these features and at [Starting to Develop Visual Studio Extensions](#).

Installing the Visual Studio SDK

The Visual Studio SDK is an optional feature in Visual Studio setup. You can also install the VS SDK later on. For more information, see [Installing the Visual Studio SDK](#).

What's New in the Visual Studio 2017 SDK

The Visual Studio SDK has some new features such as the VSIX v3 format as well as breaking changes which may require you to update your extension. For more information, see [What's New in the Visual Studio 2017 SDK](#).

Visual Studio User Experience Guidelines

Get great tips for designing the UI for your extension in [Visual Studio User Experience Guidelines](#).

You can also learn how to make your extension look great on high DPI devices with our [Addressing DPI Issues](#) topic.

Take advantage of the [Image Service and Catalog](#) for great image management and support for high DPI and theming.

Finding and Installing Existing Visual Studio Extensions

You can find Visual Studio extensions in the **Extensions and Updates** dialog on the **Tools** menu. For more information, see [Finding and Using Visual Studio Extensions](#). You can also find extensions in the [Visual Studio Marketplace](#)

Visual Studio SDK Reference

You can find the Visual Studio SDK API reference at [Visual Studio SDK Reference](#).

Visual Studio SDK Samples

You can find open source examples of VS SDK extensions on GitHub at [Visual Studio Samples](#). This GitHub repo contains samples that illustrate various extensible features in Visual Studio.

Other Visual Studio SDK Resources

If you have questions about the VSSDK or want to share your experiences developing extensions, you can use the [Visual Studio Extensibility Forum](#) or the [ExtendVS Gitter Chatroom](#).

You can find more information in the [VSX Arcana blog](#) and a number of blogs written by Microsoft MVPs:

- [Favorite Visual Studio Extensions](#)
- [Visual Studio Extensibility](#)
- [Extending Visual Studio](#)

See Also

[Creating an Extension with a Menu Command](#)

[How to: Migrate Extensibility Projects to Visual Studio 2017](#)

[FAQ: Converting Add-ins to VSPackage Extensions](#)

[Managing Multiple Threads in Managed Code](#)

[Extending Menus and Commands](#)

[Adding Commands to Toolbars](#)

[Extending and Customizing Tool Windows](#)

[Editor and Language Service Extensions](#)

[Extending Projects](#)

[Extending User Settings and Options](#)

[Creating Custom Project and Item Templates](#)

[Extending Properties and the Property Window](#)

[Extending Other Parts of Visual Studio](#)

[Using and Providing Services](#)

[Managing VSPackages](#)

[Visual Studio Isolated Shell](#)

[Shipping Visual Studio Extensions](#)

[Inside the Visual Studio SDK](#)

[Support for the Visual Studio SDK](#)

[Archive](#)

[Visual Studio SDK Reference](#)

Analyze and model your architecture

3/27/2018 • 3 min to read • [Edit Online](#)

Make sure your app meets architectural requirements by using Visual Studio architecture and modeling tools to design and model your app.

- Understand existing program code more easily by using Visual Studio to visualize the code's structure, behavior, and relationships.
- Educate your team in the need for respecting architectural dependencies.
- Create models at different levels of detail throughout the application lifecycle as part of your development process.

See [Scenario: Change your design using visualization and modeling](#).

To

| | |
|---|---|
| <p>Visualize code:</p> <ul style="list-style-type: none">- See the code's organization and relationships by creating code maps. Visualize dependencies between assemblies, namespaces, classes, methods, and so on.- See the class structure and members for a specific project by creating class diagrams from code.- Find conflicts between your code and its design by creating dependency diagrams to validate code. | <ul style="list-style-type: none">- Visualize code- Working with Classes and Other Types (Class Designer)- Video: Understand design from code with Visual Studio 2015 code maps- Video: Validate your architecture dependencies in real time |
| <p>Define the architecture:</p> <ul style="list-style-type: none">- Define and enforce constraints on dependencies between the components of your code by creating dependency diagrams. | <ul style="list-style-type: none">- Video: Validate architecture dependencies with Visual Studio (Channel 9) |
| <p>Validate your system with the requirements and intended design:</p> <ul style="list-style-type: none">- Validate code dependencies with dependency diagrams that describe the intended architecture and prevent changes that might conflict with the design. | <ul style="list-style-type: none">- Video: Validate architecture dependencies with Visual Studio (Channel 9) |
| <p>Share models, diagrams, and code maps using Team Foundation version control:</p> <ul style="list-style-type: none">- Put code maps, projects, and dependency diagrams under Team Foundation version control so you can share them. | |
| <p>Customize models and diagrams:</p> <ul style="list-style-type: none">- Create your own domain-specific languages. | <ul style="list-style-type: none">- Modeling SDK for Visual Studio - Domain-Specific Languages |

| | |
|--|---|
| Generate text using T4 templates: | - Code Generation and T4 Text Templates |
| <ul style="list-style-type: none"> - Use text blocks and control logic inside templates to generate text-based files. - T4 template build with MSBuild included in Visual Studio | |

To see which versions of Visual Studio support each feature, see [Version support for architecture and modeling tools](#)

Types of Models and Their Uses

MODEL TYPE AND TYPICAL USES

Code maps

Code maps help you see the organization and relationships in your code.

Typical uses:

- Examine program code so you can better understand its structure and its dependencies, how to update it, and estimate the cost of proposed changes.

See:

- [Map dependencies across your solutions](#)
- [Use code maps to debug your applications](#)
- [Find potential problems using code map analyzers](#)

Dependency diagram

Dependency diagrams let you define the structure of an application as a set of layers or blocks with explicit dependencies. You can run validation to discover conflicts between dependencies in the code and dependencies described on a dependency diagram.

Typical uses:

- Stabilize the structure of the application through numerous changes over its life.
- Discover unintentional dependency conflicts before checking in changes to the code.

See:

- [Create dependency diagrams from your code](#)
- [Dependency Diagrams: Reference](#)
- [Validate code with dependency diagrams](#)

Domain-specific language (DSL)

A DSL is a notation that you design for a specific purpose. In Visual Studio, it is usually graphical.

Typical uses:

- Generate or configure parts of the application. Work is required to develop the notation and tools. The result can be a better fit to your domain than a UML customization.
- For large projects or in product lines where the investment in developing the DSL and its tools is returned by its use in more than one project.

See:

- [Modeling SDK for Visual Studio - Domain-Specific Languages](#)

Where can I get more information?

[Visual Studio Visualization & Modeling Tools Forum](#)

See Also

[What's new](#)

[DevOps and Application Lifecycle Management](#)

Personalize the Visual Studio IDE

1/23/2018 • 1 min to read • [Edit Online](#)

You can personalize Visual Studio in various ways to best support your own development style and requirements. Many of your settings roam with you across Visual Studio instances—see [Synchronized settings](#). This topic briefly describes different personalizations and where you can find more information.

General environment options

Many personalization options are exposed through the [Environment Options](#) dialog box. There are two ways to access this dialog box:

- On the menu bar, choose **Tools**, **Options**, and if it's not already expanded, expand the **Environment** node.
- Type `environment` in the **Quick Launch** box and choose **Environment --> General** from the results list.

TIP

When the dialog box appears, you can press **F1** for help on the various settings on that page.

Environment color themes

To change the color theme between light, dark and blue, type `environment` in the **Quick Launch** box, and then choose **Environment --> General**. In the **Options** dialog box, change the **Color theme** option.

To change colorization options in the editor, type `environment` in the **Quick Launch** box, and then choose **Environment --> Fonts and Colors**. See [How to: Change fonts and colors](#).

Main menu casing

You can change the main menu casing between **Title Case** ("File") and **All Caps** ("FILE"). Type `environment` in the **Quick Launch** box, select **Environment --> General**, and then change the **Apply title case styling to menu bar** option.

Customizing menus and toolbars

To add or remove menu or toolbar items, see [How to: Customize menus and toolbars](#).

Start page

To create a custom start page for you and your team, see [Customizing the Start page](#).

Window layouts

You can define and save multiple window layouts and switch between them. For example, you can define one layout for coding and one for debugging. To arrange window positions and behavior and save custom layouts, see [Customizing window layouts](#).

External tools

You can customize the **Tools** menu to launch external tools. For more information, see [Managing external](#)

tools.

See also

[Visual Studio IDE overview](#)

[Quickstart: First look at the Visual Studio IDE](#)

How to: Change Fonts and Colors in Visual Studio

3/12/2018 • 1 min to read • [Edit Online](#)

You can customize the color of the IDE frame and tool windows in Visual Studio in several ways.

TIP

For information about how to change the colors of the code editor, see [How to: Change Fonts and Colors in the Editor](#).

Change the Color Theme of the IDE

1. On the menu bar, choose **Tools, Options**.
2. In the options list, choose **Environment, General**.
3. In the **Color theme** list, choose either the default **Blue** theme, **Dark** or **Light**.

NOTE

When you change a color theme, text in the IDE reverts to the default or previously customized fonts and sizes.

TIP

You can create and edit Visual Studio themes by installing the [Visual Studio Color Theme Editor](#).

Use Windows High Contrast Colors

Choose the **Left Alt + Left Shift + PrtScn** keys.

WARNING

This option sets high contrast for all applications and UI on the current computer.

Change IDE Fonts

You can change the font and text size for all windows and dialog boxes in the IDE. You can choose to customize only certain windows and other text elements.

To change the font and size of all text in the IDE

1. On the menu bar, choose **Tools, Options**.
2. In the options list, choose **Environment, Fonts and Colors**.
3. In the **Show settings for** list, choose **Environment Font**.

TIP

If you want to change the font for tool windows only, in the **Show settings for** list, choose **[All Text Tool Windows]**.

4. In the **Font** list, choose a font.
5. In the **Size** list, choose a text size, and then choose the **OK** button.

See also

[Accessibility Features of Visual Studio How to: Change Fonts and Colors in the Editor](#)

How to: Customize Menus and Toolbars in Visual Studio

12/22/2017 • 2 min to read • [Edit Online](#)

You can customize Visual Studio not only by adding and removing toolbars and menus on the menu bar, but also by adding and removing commands on any given toolbar or menu.

WARNING

After you customize a toolbar or menu, make sure that its check box remains selected in the **Customize** dialog box. Otherwise, your changes won't persist after you close and reopen Visual Studio.

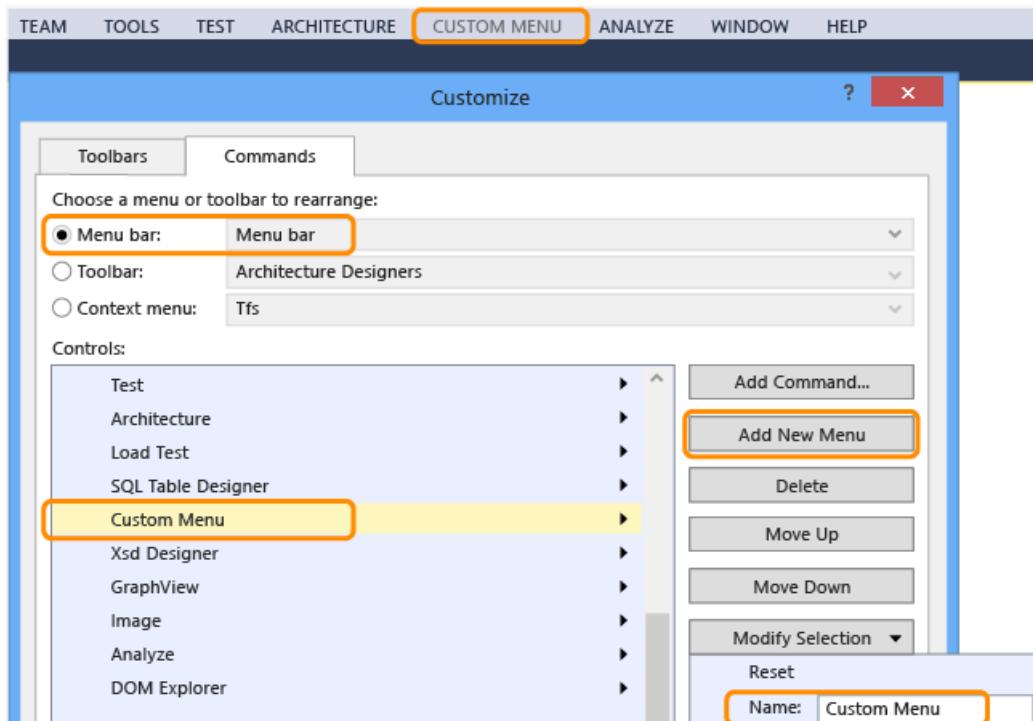
Adding, removing, or moving a menu on the menu bar

1. On the menu bar, choose **Tools, Customize**.

The **Customize** dialog box opens.

2. On the **Commands** tab, leave the **Menu bar** option button selected, leave **Menu Bar** selected in the list next to that option, and then perform one of the following sets of steps:

- To add a menu, choose the **Add New Menu** button, choose the **Modify Selection** button, and then name the menu that you want to add.



- To remove a menu, choose it in the **Controls** list, and then choose the **Delete** button.
- To move a menu within the menu bar, choose the menu in the **Controls** list, and then choose the **Move Up** or **Move Down** button.

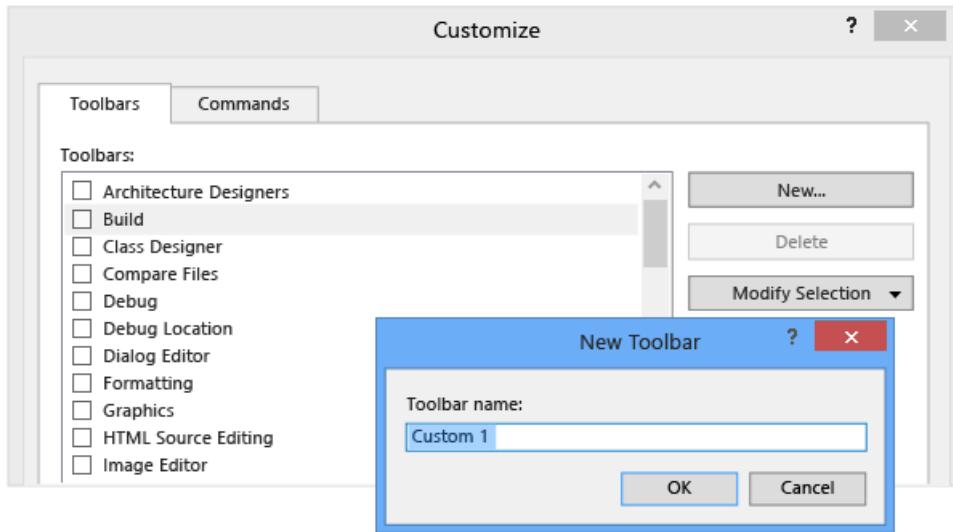
Adding, removing, or moving a toolbar

1. On the menu bar, choose **Tools, Customize**.

The **Customize** dialog box opens.

2. On the **Toolbar** tab, perform one of the following sets of steps:

- To add a toolbar, choose the **New** button, specify a name for the toolbar that you want to add, and then choose the **OK** button.



- To remove a custom toolbar, choose it in the **Toolbars** list, and then choose the **Delete** button.

IMPORTANT

You can delete toolbars that you create but not default toolbars.

- To move a toolbar to a different docking location, choose it in the **Toolbars** list, choose the **Modify Selection** button, and then choose a location in the list that appears.

You can also drag a toolbar by its left edge to move it anywhere in the main docking area.

NOTE

For more information about how to improve the usability and accessibility of toolbars, see [How to: Set IDE Accessibility Options](#).

Customizing a menu or a toolbar

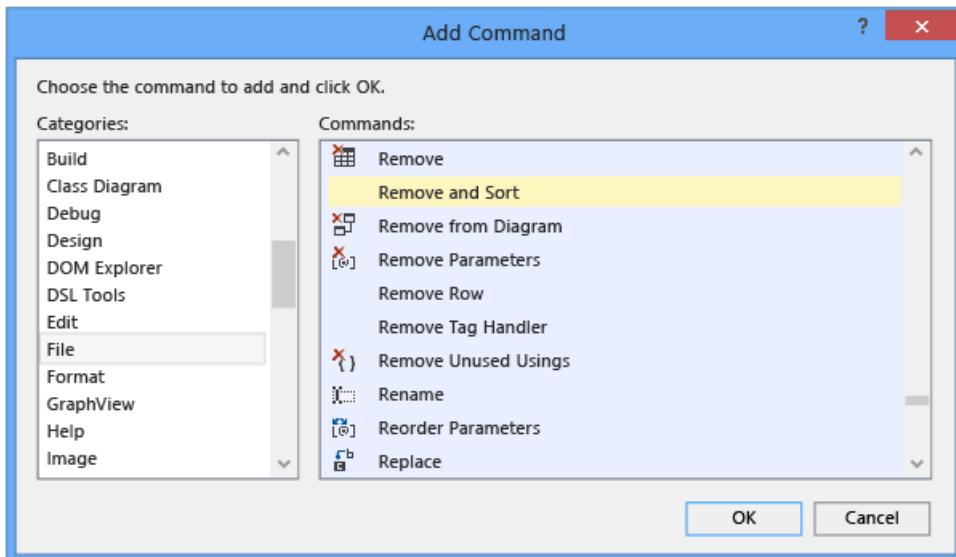
1. On the menu bar, choose **Tools, Customize**.

The **Customize** dialog box opens.

2. On the **Commands** tab, choose the option button for the type of element that you want to customize.
3. In the list for that type of element, choose the menu or toolbar that you want to customize, and then perform one of the following sets of steps:

- To add a command, choose the **Add Command** button.

In the **Add Command** dialog box, choose an item in the **Categories** list, choose an item in the **Commands** list, and then choose the **OK** button.



- To delete a command, choose it in the **Controls** list, and then choose the **Delete** button.
- To reorder commands, choose a command in the **Controls** list, and then choose the **Move Up** or **Move Down** button.
- To group commands under a horizontal line, choose the first command in the **Controls** list, choose the **Modify Selection** button, and then choose **Begin a Group** in the menu that appears.

Resetting a menu or a toolbar

1. On the menu bar, choose **Tools, Customize**.

The **Customize** dialog box opens.

2. On the **Commands** tab, choose the option button for the type of element that you want to reset.
3. In the list for that type of element, choose the menu or toolbar that you want to reset.
4. Choose the **Modify Selection** button, and then choose **Reset** in the menu that appears.

You can also reset all menus and toolbars by choosing the **Reset All** button.

See also

[Personalizing the IDE](#)

[Customizing the editor](#)

Customize window layouts in Visual Studio

1/26/2018 • 9 min to read • [Edit Online](#)

In Visual Studio you can customize the position, size and behavior of windows to create window layouts that work best for various development workflows. When you customize the layout, the IDE remembers it. For example, if you change the docking location of **Solution Explorer** and then close Visual Studio, the next time that you start, even if you are working on another computer, **Solution Explorer** will be docked in that same location. You can also give a custom layout a name and save it, and then switch between layouts with a single command. For example you could create a layout for editing, and another for debugging, and switch between them by using the **Window | Apply Window Layout** menu command.

Kinds of windows

Tool and document windows

The IDE has two basic window types, *tool windows* and *document windows*. Tool windows include Solution Explorer, Server Explorer, Output Window, Error List, the designers, the debugger windows, and so on. Document windows contain source code files, arbitrary text files, config files, and so on. Tool windows can be resized and dragged by their title bar. Document windows can be dragged by their tab. Right-click on the tab or title bar to set other options on the window.

The **Window** menu shows options for docking, floating and hiding windows in the IDE. Right click on a window tab or title bar to see additional options for that specific window. You can display more than one instance of certain tool windows at a time. For example, you can display more than one web browser window, and you can create additional instances of some tool windows by choosing **New Window** on the **Window** menu.

Preview tab (document windows)

In the Preview tab, you can view files in the editor without opening them. You can preview files by choosing them in **Solution Explorer**, during debugging when you step into files, with Go to Definition, and when you browse through results of a search. Preview files appear in a tab on the right side of the document tab well. The file opens for editing if you modify it or choose the **Open**.

Tab groups

Tab Groups extend your ability to manage limited workspace while you are working with two or more open documents in the IDE. You can organize multiple document windows and tool windows into either vertical or horizontal Tab Groups and shuffle documents from one Tab Group to another.

Split windows

When you have to view or edit two locations at once in a document, you can split windows. To divide your document into two independently scrolling sections, click **Split** on the **Window** menu. Click **Remove Split** on the **Window** menu to restore the single view.

Toolbars

Toolbars can be arranged by dragging, or by using the **Customize** dialog box. For more information about how to position and customize toolbars, see [How to: Customize Menus and Toolbars](#).

Arrange and dock Windows

Both document windows and tool windows can be *docked*, so that it has a position and size within the IDE window frame, or floating as a separate window independent of the IDE. Tool windows can be docked anywhere inside the IDE frame; some tool windows can be docked as tabbed windows in the editor frame. Document

windows can be docked within the editor frame, and they can be pinned to their current position in the tab order. You can dock multiple windows to float together in a "raft" over or outside of the IDE. Tool windows can also be hidden or minimized.

You can arrange windows in the following ways:

- Pin document windows to the left of the tab well.
- Tab-dock windows to the editing frame.
- Dock tool windows to the edge of a frame in the IDE.
- Float document or tool windows over or outside the IDE.
- Hide tool windows along the edge of the IDE.
- Display windows on different monitors.
- Reset window placement to the default layout or to a saved custom layout.

Tool and document windows can be arranged by dragging, by using commands on the **Window** menu, and by right-clicking the title bar of the window to be arranged.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalize the Visual Studio IDE](#).

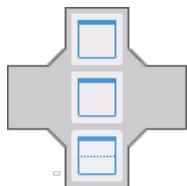
Dock Windows

When you click and drag the title bar of a tool window, or the tab of document window, a guide diamond appears. During the drag operation, when the mouse cursor is over one of the arrows in the diamond, a shaded area will appear that shows you where the window will be docked if you release the mouse button now.

To move a dockable window without snapping it into place, choose the **Ctrl** key while you drag the window.

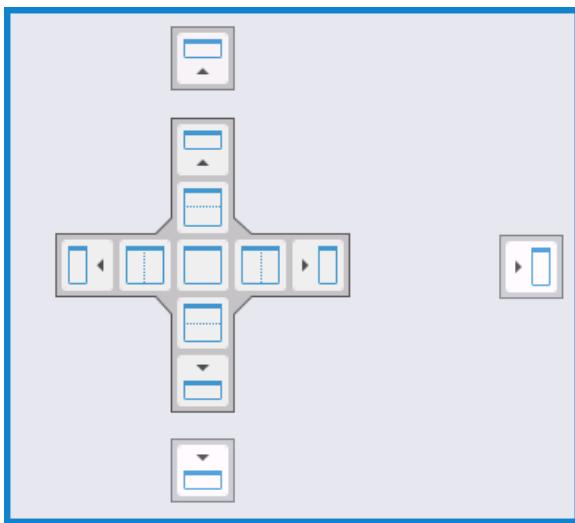
To return a tool window or document window to its most recent docked location, press **CTRL** while you double-click the title bar or tab of the window.

The following illustration shows the guide diamond for document windows, which can only be docked within the editing frame:

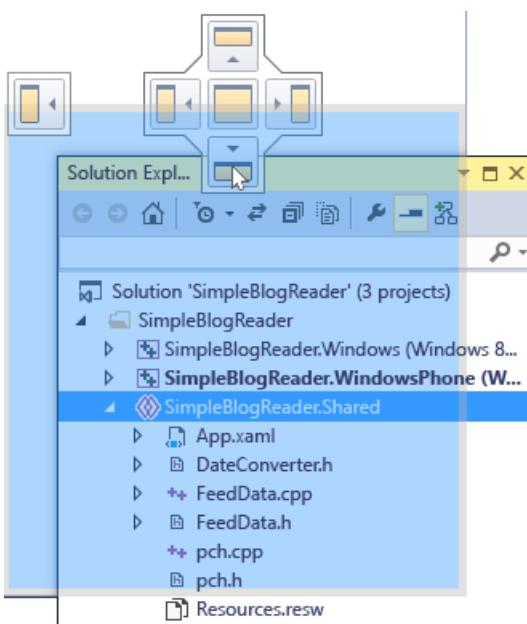


Tool windows can be fastened to one side of a frame in the IDE or within the editing frame. A guide diamond appears when you drag a tool window to another location to help you to easily re-dock the window.

Guide diamond for tool windows

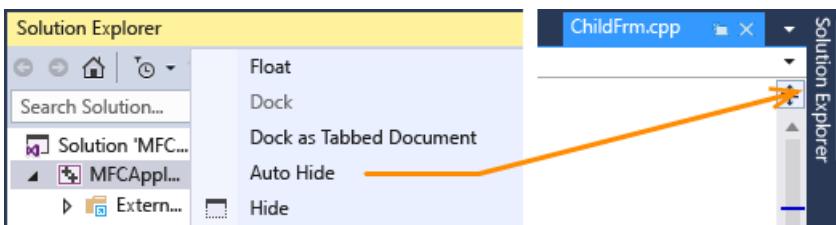


The following illustration shows Solution Explorer being docked in a new location, which is shown by the blue shaded area:



Close and auto-hide tool windows

You can close a tool window by clicking the X in the upper right of the title bar; to reopen the window, use its keyboard shortcut or menu command. Tool windows support a feature named Auto Hide, which causes a window to slide out of the way when you use a different window. When a window is auto-hidden, its name appears on a tab at the edge of the IDE. To use the window again, point to the tab so that the window slides back into view.



NOTE

To set whether Auto Hide operates on tool windows individually or as docked groups, select or clear **Auto Hide button affects active tool windows only** in the **Options** dialog box. For more information, see [General, Environment, Options Dialog Box](#).

NOTE

Tool windows that have Auto Hide enabled may temporarily slide into view when the window has focus. To hide the window again, select an item outside of the current window. When the window loses focus, it slides back out of view.

Specifying a second monitor

If you have a second monitor and your operating system supports it, you can choose which monitor displays a window. You can even group multiple windows together in "rafts" on other monitors.

TIP

You can create multiple instances of **Solution Explorer** and move them to another monitor. Right-click the window and choose **New Solution Explorer View**. You can return all windows back to the original monitor by double-clicking while choosing the Ctrl key.

Reset, name, and switch between window layouts

You can return the IDE to the original window layout for your settings collection by using the **Reset Window Layout** command. When you run this command, the following actions occur:

- All windows are moved to their default positions.
- Windows that are closed in the default window layout are closed.
- Windows that are open in the default window layout are opened.

Create and save custom layouts

Visual Studio enables you to save up to 10 custom window layouts and quickly switch between them. The following steps show how to create, save, invoke, and manage custom layouts that take advantage of multiple monitors with both docked and floating tool windows.

First, create a test solution that has two projects, each with a different optimal layout.

Create a UI project and customize the layout

1. In the **New Project** dialog, create a C# WPF Desktop Application and call it whatever you like. Pretend that this is the project where we'll be working on the user interface, so we want to maximize the space for the designer window and move other tool windows out of the way.
2. If you have multiple monitors, pull the **Solution Explorer** window and the **Properties** window over to your second monitor. On a single monitor system, try closing all the windows except the designer.
3. Press **Ctrl + Alt + X** to display the Toolbox. If the window is docked, drag it so that it floats somewhere where you'd like to position it, on either monitor.
4. Press F5 to put Visual Studio into debugging mode. Adjust the position of the Autos, Call Stack and Output debugging windows the way you want them. The layout you are about to create will apply to both editing mode and debugging mode.
5. When your layouts in both debugging mode and editing mode are how you want them, from the main menu choose **Window > Save Window Layout**. Call this layout "Designer."

Note that your new layout is assigned the next Keyboard shortcut from the reserved list of Ctrl + Alt + 1...0.

Create a database project and layout

1. Add a new **SQL Server Database** project to the solution.
2. Right-click on the new project in Solution Explorer and choose **View in Object Explorer**. This displays the

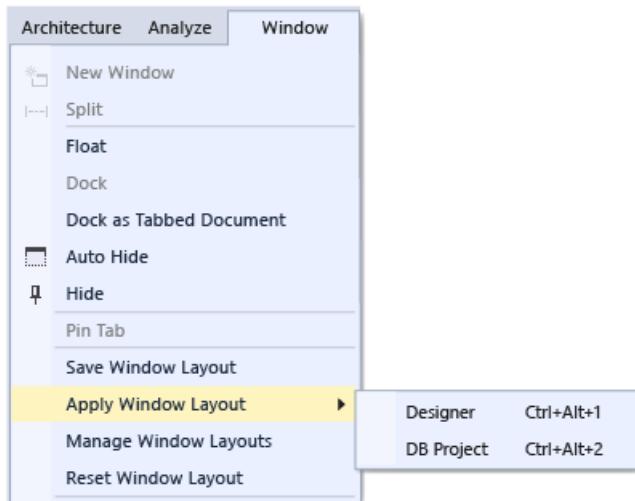
SQL Server Object Explorer window, which enables you to access tables, views and other objects in your database. You can either float this window or leave it docked. Adjust the other tool windows the way you want them. For added realism, you can add an actual database, but it's not necessary for this walkthrough.

- When your layout is how you want it, from the main menu choose **Window > Save Window Layout**.

Call this layout "DB Project." (We won't bother with a debug mode layout for this project.)

Switch between the layouts

- To switch between layouts, use the Keyboard shortcuts, or from the main menu choose **Window > Apply Window Layout**.



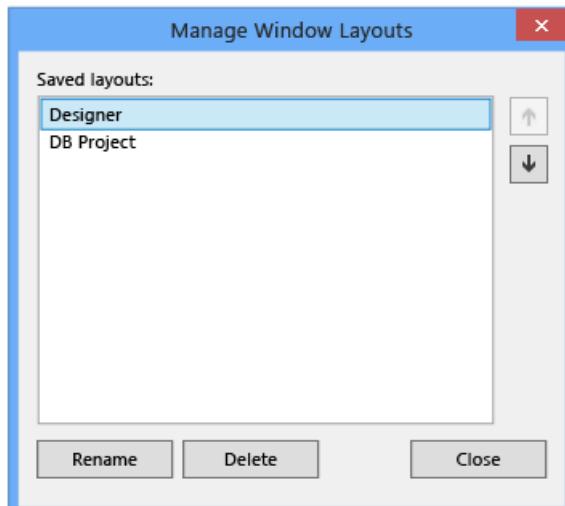
After applying the UI layout, note how the layout is preserved both in editing mode and in debug mode.

If you have a multi monitor setup at work and a single monitor laptop at home, you can create layouts that are optimized for each machine.

Note: If you apply a multi-monitor layout on a single-monitor system, the floating windows that you placed on the second monitor will now be hidden behind the Visual Studio window. You can bring these windows to the front by pressing Alt + Tab. If you later open Visual Studio with multiple monitors, you can restore the windows to their specified positions by re-applying the layout.

Manage and roam your layouts

- You can remove, rename or reorder your custom layout by choosing **Window > Manage Window Layouts**. If you move a layout, the key binding is automatically adjusted to reflect the new position in the list. The bindings cannot be otherwise modified, and so you can store a maximum of 10 layouts at a time.



To remind yourself which keyboard shortcut is assigned to which layout, choose **Window > Apply Window Layout**.

These layouts automatically roam between Visual Studio editions, and also between Blend instances on separate machines, and from any Express edition to any other Express organization. However, layouts do not roam across Visual Studio, Blend and Express.

Related Topics

[How to: Move Around in the IDE](#)

Customize the start page for Visual Studio

2/8/2018 • 1 min to read • [Edit Online](#)

You can customize the startup experience for Visual Studio in several different ways, such as showing the **Open Project** dialog box or opening the solution that was loaded most recently. You can also show a custom start page, which is a Windows Presentation Foundation (WPF) XAML page that runs in a tool window and can run commands that are internal to Visual Studio.

To change the startup item

1. On the menu bar, choose **Tools**, **Options**.
2. Expand **Environment**, and then choose **Startup**.
3. In the **At startup** list, choose the item to be displayed after Visual Studio launches.

To show a custom start page

You can [create your own custom start page](#) using the Visual Studio SDK, or use one that somebody else has already created. For example, you can find custom start pages at the [Visual Studio Marketplace](#).

To install a custom start page, open the .vsix file, or copy and paste the start page files into the **%USERPROFILE%\Documents\Visual Studio 2017\StartPages** folder on your computer.

To select which custom start page to display

1. On the menu bar, choose **Tools**, **Options**.
2. Expand **Environment**, and then choose **Startup**.
3. In the **Customize Start Page** list, choose the page that you want.

NOTE

If an error in a custom start page causes Visual Studio to crash, you can start Visual Studio in safe mode and then set it to use the default start page. See [/SafeMode \(devenv.exe\)](#).

See also

[Personalize the Visual Studio IDE](#)

Find and use Visual Studio Extensions

3/5/2018 • 7 min to read • [Edit Online](#)

Visual Studio extensions are code packages that run inside Visual Studio and provide new or improved Visual Studio features. You can find more information about Visual Studio extensions here: [Visual Studio SDK](#).

You can use the **Extensions and Updates** dialog box to install Visual Studio extensions and samples from websites and other locations, and then enable, disable, update, or uninstall them. (**Tools / Extensions and Updates**, or type **Extensions** in the **Quick Launch** window). The dialog box also shows updates for installed samples and extensions. You can also download extensions from websites, or get them from other developers.

NOTE

Starting in Visual Studio 2015, extensions hosted on the Visual Studio Marketplace are automatically updated. You can change this setting through the **Extensions and Updates** dialog. See the section on **Automatic Extension Updates** below for details.

Finding Visual Studio Extensions

You can install extensions from the [Visual Studio Marketplace](#). Extensions may be controls, samples, templates, tools, or other components that add functionality to Visual Studio. Visual Studio supports extensions in the VSIX package format—these include project templates, item templates, **Toolbox** items, Managed Extension Framework (MEF) components, and VSPackages. You can also download and install MSI-based extensions, but the **Extensions and Updates** dialog box can't enable or disable them. The Visual Studio Marketplace contains both VSIX and MSI extensions.

Installing or Uninstalling Visual Studio Extensions

In the **Extensions and Updates**, find the extension you want to install. (If you know the name or part of the name of the extension, you can search in the **Search** window.) Click **Download**. The extension will be scheduled for install. Your extension will be installed once all instances of Visual Studio are closed.

If you try to install an extension that has dependencies, the installer verifies whether they're already installed. If they aren't installed, the **Extensions and Updates** dialog box lists the dependencies that must be installed before you can install the extension.

If you want to stop using an extension, you can either disable it or uninstall it. Disabling an extension keeps it installed but unloaded. You can disable only VSIX extensions; extensions that were installed using an MSI can only be uninstalled. Find the extension and click **Uninstall** or **Disable**. You must restart Visual Studio in order to unload a disabled extension.

Per-User and Administrative Extensions

Most extensions are per-user extensions and are installed in the `%LocalAppData%\Microsoft\VisualStudio\<Visual Studio version>\Extensions\` folder. A few extensions are administrative extensions, and are installed in the `<Visual Studio installation folder>\Common7\IDE\Extensions\` folder.

To protect your system against extensions that may contain errors or malicious code, you can restrict per-user extensions to load only when Visual Studio is run with normal user permissions. This means that per-user extensions are disabled when Visual Studio is run with administrative user permissions. To do this, go to the

Extensions and Updates options page (**Tools / Options, Environment, Extensions and Updates**, or just type **Extension** in the **Quick Launch** window). Clear the **Load per user extensions when running as administrator** check box, then restart Visual Studio.

Automatic Extension Updates

Per-user extensions are automatically updated when a new version is available for the Visual Studio Marketplace. The new version of the extension is detected and installed in the background and on the next restart of Visual Studio, the new version of the extension will be running.

Only per-user extensions can be automatically updated. Administrative extensions which are installed for all users will not be updated and you still manually install new versions through the **Extensions and Updates** dialog's **Updates** node. You can see which extensions will be automatically updated in the extension's details pane of **Extensions and Updates** dialog.

If you wish to disable automatic updates, you can disable the feature for all extensions or only specific extensions.

- To disable automatic updates for all extensions, choose the **Change your Extensions and Updates settings** link in the **Extensions and Updates** dialog and uncheck **Automatically update extensions**.
- To disable automatic updates for a specific extension, uncheck the **Automatically update this extension** option in the extension's details pane on the right side of the **Extensions and Updates** dialog.

NOTE

Starting in Visual Studio 2015 Update 2, you can specify (in **Tools / Options / Environment / Extensions and Updates**) whether you want automatic updates for per-user extensions, all user extensions or both (the default setting).

Extension Crash/Unresponsiveness Notifications

New in **Visual Studio 2017 version 15.3**, Visual Studio notifies you if it suspects that an extension was involved in a crash during a previous session. When Visual Studio crashes, it stores the exception stack. The next time Visual Studio launches, it examines the stack, starting with the leaf and working towards the base. If Visual Studio determines that a frame belongs to a module that is part of an installed and enabled extension, it shows a notification.

New in **Visual Studio 2017 version 15.6**, Visual Studio also notifies you if it suspects an extension is causing the UI to be unresponsive.

When these notifications are shown, you can ignore the notification or take one of the following actions:

- Choose **Disable this extension**. Visual Studio disables the extension and lets you know whether you need to restart your system for the disabling to take effect. You can re-enable the extension in the **Extensions and Updates** dialog box if you want.
- Choose **Never show this message again**.
 - If the notification concerns a crash in a previous session, Visual Studio will no longer show a notification when a crash associated with this extension occurs. Visual Studio will still show notifications when unresponsiveness can be associated with this extension, or for crashes or unresponsiveness that can be associated with other extensions.
 - If the notification concerns unresponsiveness, the IDE will no longer show a notification when this extension is associated with unresponsiveness. Visual Studio will still show crash-related notifications for this extension, and crash- and unresponsiveness-related notifications for other extensions.
- Choose **Learn more** to come to this page.

- Choose the **X** button at the end of the notification to dismiss the notification. A new notification will appear for future instances of the extension being associated with a crash or UI unresponsiveness.

NOTE

A UI unresponsiveness or crash notification means only that one of the extension's modules was on the stack when the UI was unresponsive or when the crash occurred. It does not necessarily mean that the extension itself was the culprit. It is possible that the extension called code which is part of Visual Studio, which in turn resulted in unresponsive UI or a crash. However, the notification may still be useful if the extension which led to the UI unresponsiveness or crash is not important to you. In this case, disabling the extension avoids the UI unresponsiveness or the crash in the future, without impacting your productivity.

Sample Master Copies and Working Copies

When you install an online sample, the solution is stored in two locations:

- A working copy is stored in the location that you specified in the **New Project** dialog box.
- A separate master copy is stored on your computer.

You can use the **Extensions and Updates** dialog box to perform these samples-related tasks:

- List the master copies of samples that you have installed.
- Disable or uninstall the master copy of a sample.
- Install Sample Packs, which are collections of samples that relate to a technology or feature.
- Install individual online samples. (You can also do this in the **New Project** dialog box.)
- View update notifications when source code changes are published for installed samples.
- Update the master copy of an installed sample when there is an update notification.

Installing Without Using the Extensions and Updates Dialog Box

Extensions that have been packaged in .vsix files may be available in locations other than the Visual Studio Marketplace. The **Extensions and Updates** dialog box can't detect these files, but you can install a .vsix file by double-clicking the file, or selecting the file and pressing the ENTER key. After that, just follow the instructions. When the extension is installed, you can use the **Extensions and Updates** dialog box to enable it, disable it, or uninstall it.

Extension Types Not Supported by the Extensions and Updates Dialog Box

Visual Studio continues to support extensions that are installed by the Microsoft Installer (MSI) but not through the **Extensions and Updates** dialog box without modification.

TIP

If an MSI-based extension includes an extension.vsixmanifest file, the extension will appear in the **Extensions and Updates** dialog box.

Manage external tools

2/6/2018 • 2 min to read • [Edit Online](#)

You can call external tools from inside Visual Studio by using the **Tools** menu. A few default tools are available from the **Tools** menu, and you can customize the menu by adding other executables of your own.

Tools available on the Tools menu

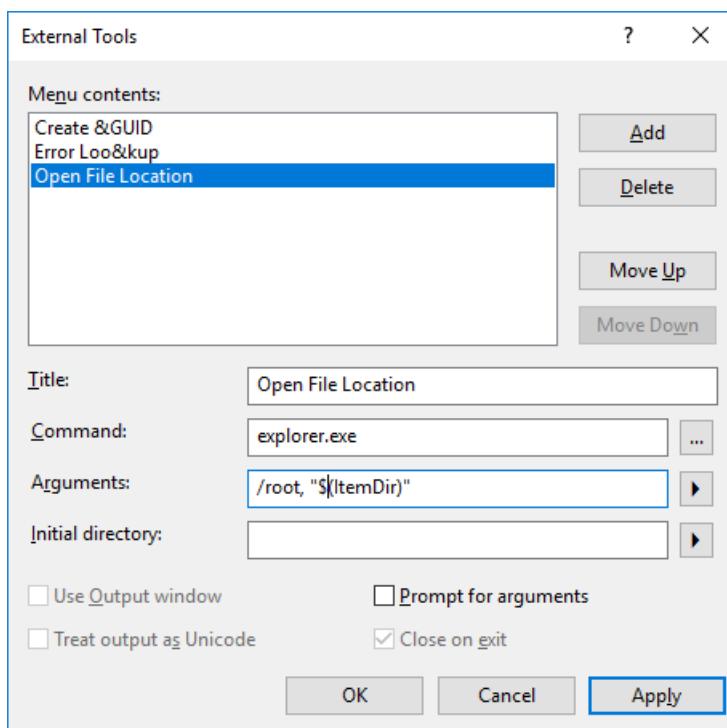
The **Tools** menu contains several built-in commands, such as:

- **Extensions and Updates** for [managing Visual Studio Extensions](#)
- **Code Snippets Manager...** for [organizing Code Snippets](#)
- **PreEmptive Protection - Dotfuscator** for launching Dotfuscator Community Edition (CE) if it is [installed](#)
- **Customize...** for [customizing menus and toolbars](#)
- **Options...** for [setting a variety of different options for the Visual Studio IDE and other tools](#)

Add new tools to the Tools menu

You can add an external tool to appear on the **Tools** menu.

1. Open the **External Tools** dialog box by choosing **Tools > External Tools....**
2. Click **Add**, and then fill in the information. For example, the following entry causes Windows Explorer to open at the directory of the file you currently have open in Visual Studio:
 - Title: `Open File Location`
 - Command: `explorer.exe`
 - Arguments: `/root, "$(ItemDir)"`



The following is a full list of arguments that can be used when defining an external tool:

| NAME | ARGUMENT | DESCRIPTION |
|--------------------|-----------------------------------|---|
| Item Path | <code>\$(ItemPath)</code> | The complete file name of the current file (drive + path + file name). |
| Item Directory | <code>\$(ItemDir)</code> | The directory of the current file (drive + path). |
| Item File Name | <code>\$(ItemFilename)</code> | The file name of the current file (file name). |
| Item Extension | <code>\$(ItemExt)</code> | The file name extension of the current file. |
| Current Line | <code>\$(CurLine)</code> | The current line position of the cursor in the code window. |
| Current Column | <code>\$(CurCol)</code> | The current column position of the cursor in the code window. |
| Current Text | <code>\$(CurText)</code> | The selected text. |
| Target Path | <code>\$(TargetPath)</code> | The complete file name of the item to be built (drive + path + file name). |
| Target Directory | <code>\$(TargetDir)</code> | The directory of the item to be built. |
| Target Name | <code>\$(TargetName)</code> | The file name of the item to be built. |
| Target Extension | <code>\$(TargetExt)</code> | The file name extension of the item to be built. |
| Binary Directory | <code>\$(BinDir)</code> | The final location of the binary that is being built (defined as drive + path). |
| Project Directory | <code>\$(ProjDir)</code> | The directory of the current project (drive + path). |
| Project File Name | <code>\$(ProjFileName)</code> | The file name of the current project (drive + path + file name). |
| Solution Directory | <code>\$(SolutionDir)</code> | The directory of the current solution (drive + path). |
| Solution File Name | <code>\$(SolutionFileName)</code> | The file name of the current solution (drive + path + file name). |

NOTE

The IDE status bar displays the Current Line and Current Column variables to indicate where the insertion point is located in the active Code Editor. The Current Text variable returns the text or code selected at that location.

See also

Synchronize your settings in Visual Studio

1/2/2018 • 2 min to read • [Edit Online](#)

When you sign in to Visual Studio on multiple computers using the same personalization account, by default your settings are synchronized on all computers.

Synchronized settings

By default, the following settings are synchronized.

- Development settings (You have to select a set of settings the first time you run Visual Studio, but you can change the selection anytime. For more information, see [Personalize the Visual Studio IDE](#).)
- The following options in the **Tools | Options** pages:
 - **Theme** and menu bar casing settings, on the **Environment, General** options page
 - All settings on the **Environment, Fonts and Colors** options page
 - All keyboard shortcuts, on the **Environment, Keyboard** options page
 - All settings on the **Environment, Tabs and Windows** options page
 - All settings on the **Environment, StartUp** options page
 - All settings on the **Text Editor** options pages
 - All settings on the **XAML Designer** options pages
- User-defined command aliases. For more information about how to define command aliases, see [Visual Studio Command Aliases](#).
- User-defined window layouts in **Window | Manage Window Layouts** page

Turn off synchronized settings on a particular computer

Synchronized settings for Visual Studio are turned on by default. You can turn off synchronized settings on a computer by going to the **Tools | Options | Environment | Accounts** page and unchecking the checkbox. For example, if you decide not to synchronize Visual Studio's settings on Computer A, any setting changes made on Computer A do not appear on Computer B or Computer C. Computer B and C will continue to synchronize with each other, but not with Computer A.

Synchronize settings across Visual Studio family products and editions

Settings can be synchronized across any edition of Visual Studio, including the Community edition. Settings are also synchronized across Visual Studio family products. However, each of these family products may have its own settings that are not shared with Visual Studio. For example, settings specific to one product on Computer A will be shared with another on Computer B, but not with Visual Studio on Computer A or B.

Side-by-side synchronized settings

In Visual Studio 15.3 and later, we've stopped sharing certain settings, like tool window layout, between different side-by-side installations of Visual Studio 2017 by changing the location of `currentSettings.vssettings` file in `%userprofile%\Documents\Visual Studio 2017\Settings` to an installation specific folder that is similar to

```
%localappdata%\Microsoft\VisualStudio\15.0_xxxxxxx\Settings .
```

NOTE: To use the new installation specific settings, you must complete a fresh installation. When you perform an upgrade of an existing Visual Studio 2017 installation to the most current update, it will use the existing shared location. If you currently have side-by-side installations of Visual Studio 2017 and decide to upgrade, and want to use the new installation specific settings file location, follow these steps:

1. After the upgrade, use the Import\Export settings wizard to export all our existing settings to some location outside of `%localappdata%\Microsoft\VisualStudio\15.0_xxxxxxx` folder.
2. Open the **Developer Command Prompt for VS 2017** of the upgraded Visual Studio installation and run `devenv /resetuserdata` from it.
3. Launch Visual Studio and import the saved settings from the exported settings file.

See also

[Personalizing the IDE](#)

Optimize Visual Studio performance

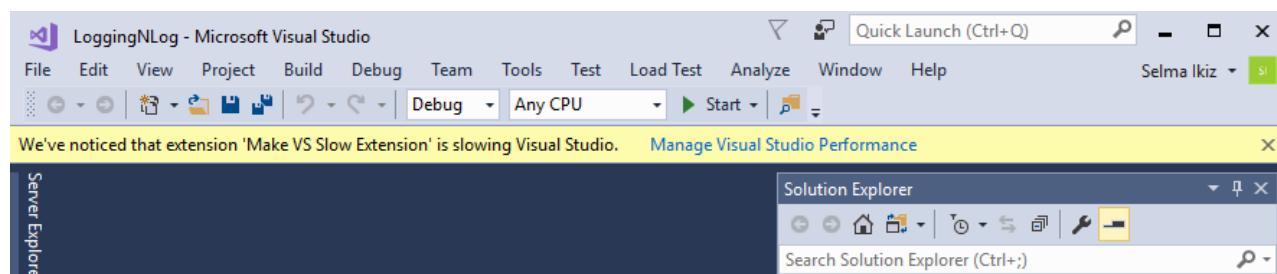
2/21/2018 • 2 min to read • [Edit Online](#)

Visual Studio is designed to start up as quickly and efficiently as possible. However, certain Visual Studio extensions and tool windows can adversely affect startup time when they are loaded. You can control the behavior of slow extensions and tool windows in the the **Manage Visual Studio Performance** dialog box. For more general tips on improving performance, see [Visual Studio performance tips and tricks](#).

Startup behavior

To avoid extending startup time, Visual Studio 2017 loads extensions using an *on demand* approach. This behavior means that extensions don't open immediately after Visual Studio starts, but on an as-needed basis. Also, because tool windows left open in a prior Visual Studio session can slow startup time, Visual Studio opens tool windows in a more intelligent way to avoid impacting startup time.

If Visual Studio detects slow startup, a pop-up message appears, alerting you to the extension or tool window that's causing the slowdown. The message provides a link to the **Manage Visual Studio Performance** dialog box. You can also access this dialog box by choosing **Help, Manage Visual Studio Performance** from the menu bar.

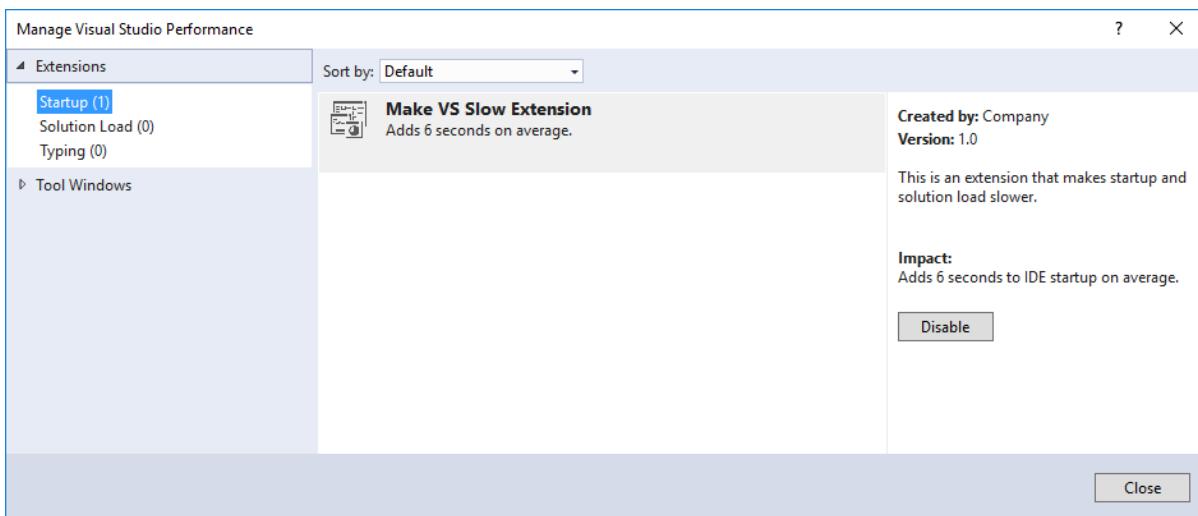


The dialog box lists the extensions and tools windows that are affecting startup performance. You can change extension and tool window settings to improve startup performance.

To change extension settings to improve startup, solution load, and typing performance

1. Open the **Manage Visual Studio Performance** dialog box by choosing **Help, Manage Visual Studio Performance** from the menu bar.

If an extension is slowing down Visual Studio startup, solution loading, or typing, the extension appears in the **Manage Visual Studio Performance** dialog box under **Extensions, Startup** (or **Solution Load** or **Typing**).



- Choose the extension you want to disable, then choose the **Disable** button.

You can always re-enable the extension for future sessions by using the Extension Manager or the Manage Visual Studio Performance dialog box.

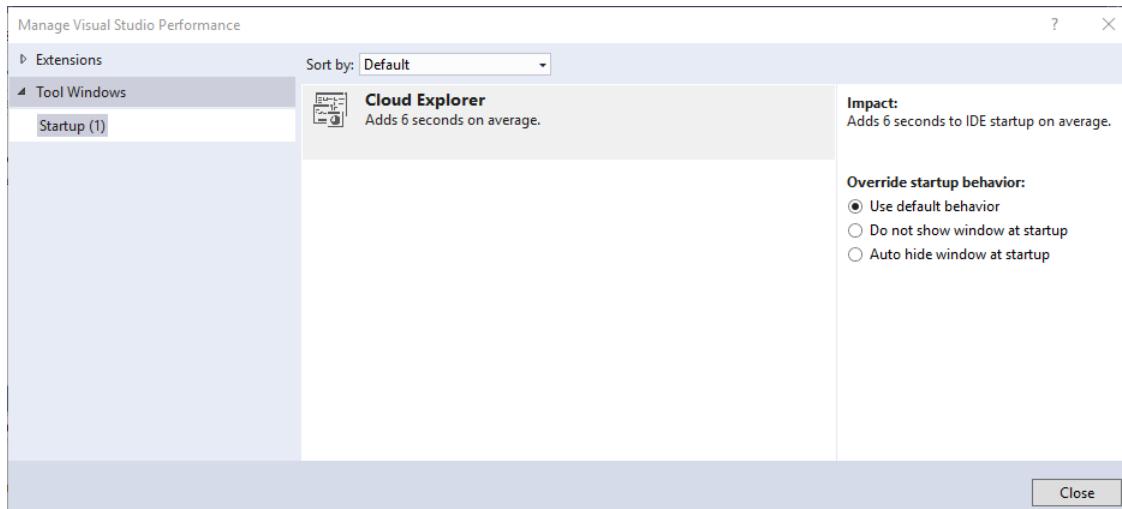
To change tool window settings to improve startup time

- Open the **Manage Visual Studio Performance** dialog box by choosing **Help, Manage Visual Studio Performance** from the menu bar.

If a tool window is slowing down Visual Studio startup, the tool window appears in the **Manage Visual Studio Performance** dialog box under **Tool Windows, Startup**.

- Choose the tool window you want to change the behavior for.
- Choose one of the following three options:

- Use default behavior:** The default behavior for the tool window. Keeping this option selected will not improve startup performance.
- Do not show window at startup:** The specified tool window is always closed when you open Visual Studio, even if you left it open in a previous session. You can open the tool window from the appropriate menu when you need it.
- Auto hide window at startup:** If a tool window was left open in a previous session, this option collapses the tool window's group at startup to avoid initializing the tool window. This option is a good choice if you use a tool window often. The tool window is still available, but no longer negatively affects Visual Studio startup time.



NOTE

Some earlier versions of Visual Studio 2017 had a feature called **lightweight solution load**. This feature is no longer available in Visual Studio 2017 version 15.5 and later. In Visual Studio 2017 version 15.5 and later, large solutions that contain managed code load much faster than previously, even without lightweight solution load.

See also

- [Visual Studio performance tips and tricks](#)

4 min to read •

Visual Studio Performance Tips and Tricks

2/7/2018 • 5 min to read • [Edit Online](#)

Visual Studio performance recommendations are intended for low memory situations, which may occur in rare cases. In these situations, you can optimize certain Visual Studio features that you may not be using. The following tips are not intended as general recommendations.

NOTE

If you're having difficulty using the product because of memory issues, let us know through the feedback tool.

Optimize your environment

- **Use a 64bit OS**

If you upgrade your system from a 32-bit version of Windows to a 64-bit version, you expand the amount of virtual memory available to Visual Studio from 2 GB to 4 GB. This enables Visual Studio to handle significantly larger workloads even though it is 32-bit process.

For more information, see [Memory limits](#) and [Using /LARGEADDRESSAWARE on 64-bit Windows](#).

Configure solution and projects

If you have a very large solution with many projects, you may benefit by making the following optimizations:

- **Unload Projects**

You can manually unload rarely used individual projects from Solution Explorer using the right-click context menu.

- **Refactor the solution**

You can split your solution into several smaller solution files with commonly used projects. This refactoring should significantly reduce memory usage for your workflow. Smaller solutions also load faster.

Configure debugging options

If you are typically running low on memory during debugging sessions, you can optimize performance by making one or more configuration changes.

- **Enable Just My Code**

The simplest optimization is to enable the **Just My Code** feature, which only loads symbols for your project. Enabling this feature can result in a significant memory saving for debugging managed applications (.NET). This option is already enabled by default in some project types.

To enable **Just My Code**, choose **Tools > Options > Debugging > General**, and then select **Enable Just My Code**.

- **Specify symbols to load**

For native debugging, loading symbol files (.pdb) is expensive in terms of memory resources. You can configure your debugger symbol settings to conserve memory. Typically, you configure the solution to only

load modules from your project.

To specify symbol loading, choose **Tools > Options > Debugging > Symbols**.

Set the options to **Only specified modules** instead of **All modules** and then specify which modules you care to load. While debugging, you can also right-click specific modules in the **Modules** window to explicitly include a module in the symbol load. (To open the window while debugging, choose **Debug > Windows > Modules**.)

For more information, see [Understanding symbol files](#).

- **Disable Diagnostic Tools**

It is recommended that you disable CPU profiling after use. This feature can consume large amounts of resources. Once CPU profiling is enabled, this state is persisted across subsequent debug sessions, so it's worth explicitly turning it off when done. You may save some resources by disabling the diagnostic tools while debugging if you do not need the provided features.

To disable the Diagnostic Tools, start a debugging session, choose **Tools > Options > Enable Diagnostic Tools**, and deselect the option.

For more information, see [Profiling Tools](#).

Disable tools and extensions

Some tools or extensions may be turned off to improve performance.

TIP

You can often isolate performance issues by turning off extensions one at a time and rechecking performance.

Managed Language Services (Roslyn)

For information about .NET Compiler Platform ("Roslyn") performance considerations, see [Performance considerations for large solutions](#).

- **Disable Full Solution Analysis**

Visual Studio performs analysis on your entire solution in order to provide a rich experience about errors before invoking a build. This feature is useful to identify errors as soon as possible. However, for very large solutions, this feature can consume significant memory resources. If you're experiencing memory pressure or similar issues, you can disable this experience to free up these resources. By default, this option is enabled for Visual Basic and disabled for C#.

To disable **Full Solution Analysis**, choose **Tools > Options > Text Editor > <Visual Basic or C#>**. Then choose **Advanced** and deselect **Enable full solution analysis**.

- **Disable CodeLens**

Visual Studio performs a **Find All References** task on each method as it is displayed. CodeLens provides features such as the inline display of the number of references. The work is performed in a separate process (for example, ServiceHub.RoslynCodeAnalysisService32). In very large solutions or on resource constrained systems, this feature can have significant impact on performance even though it is run at a low priority. If you're experiencing high CPU in this process, or memory issues (for example, when loading a large solution on a 4-GB machine), you can try disabling this feature to free up resources.

To disable CodeLens, choose **Tools > Options > Text Editor > All Languages > CodeLens**, and deselect the feature.

This feature is available in Visual Studio Professional and Visual Studio Enterprise.

Other tools and extensions

- **Disable Extensions**

Extensions are additional software components added to Visual Studio that provide new functionality or extend existing functionality. Extensions can often be a source of memory resource issues. If you're experiencing memory resource problems, try disabling extensions one at a time to see how it impacts the scenario or workflow.

To disable extensions, go to **Tools | Extensions and Updates**, and disable a particular extension.

- **Disable XAML Designer**

The XAML designer is enabled by default, but only consumes resources if you open a .XAML file. If you work with XAML files but do not wish to use the designer functionality, disable this feature to free up some memory.

To disable XAML Designer, go to **Tools > Options > XAML Designer > Enable XAML Designer**, and deselect the option.

- **Remove workloads**

You can use the Visual Studio Installer to remove workloads that are no longer used. This action can streamline the startup and runtime cost by skipping packages and assemblies that aren't needed anymore.

Force a garbage collection

The CLR uses a garbage collection memory management system. In this system, sometimes memory is used by objects that are no longer needed. This state is temporary; the garbage collector will release this memory based on its performance and resource usage heuristics. You can force the CLR to collect any unused memory by using a hotkey in Visual Studio. If there is a significant amount of garbage waiting for collection and you force a garbage collection, you should see the memory usage of the devenv.exe process drop in Task Manager. It's rarely necessary to use this method. However, after an expensive operation has completed (such as a full build, debug session, or a solution open event), it can help you determine how much memory is really being used by the process. Because Visual Studio is mixed (managed & native), it's occasionally possible for the native allocator and the garbage collector to compete for limited memory resources. Under conditions of high memory usage, it may help to force the garbage collector to run.

To force a garbage collection, use the hotkey: **Ctrl+Alt+Shift+F12, Ctrl+Alt+Shift+F12** (press it twice).

If forcing garbage collection reliably makes your scenario work, file a report through the Visual Studio feedback tool as this behavior is likely to be a bug.

For a detailed description of the CLR garbage collector, see [Fundamental of Garbage Collection](#).

See also

[Visual Studio IDE](#)

Security in Visual Studio

2/16/2018 • 1 min to read • [Edit Online](#)

You should consider security in all aspects of your application development, from design to deployment. Start by running Visual Studio as securely as possible. See [User Permissions](#).

To help you effectively develop secure applications, you should have a fundamental understanding of security concepts and the security features of the platforms for which you develop. You should also understand secure coding techniques.

Understanding Security

[Security](#)

Describes .NET Framework code access security, role-based security, security policy, and security tools.

[Defend Your Code with Top Ten Security Tips Every Developer Must Know](#)

Describes the issues that you should watch out for so that you don't compromise your data or your system.

Coding for Security

Most coding errors that result in security vulnerabilities occur because developers make incorrect assumptions when working with user input or because they don't fully understand the platform for which they're developing.

[Secure Coding Guidelines](#)

Provides guidelines for classifying your components to address security issues.

[Security Best Practices](#)

Discusses buffer overruns and the complete picture of the Microsoft Visual C++ security checks feature provided by the /GS compile-time flag.

Building for Security

Security is also an important consideration in the build process. A few additional steps can improve the security of a deployed app, and help prevent unauthorized reverse engineering, spoofing, or other attacks.

[Dotfuscator Community Edition \(CE\)](#)

Explains how to set up and start using the free PreEmptive Protection - Dotfuscator Community Edition to protect .NET assemblies from reverse-engineering and unauthorized use (such as unauthorized debugging).

[Managing Assembly and Manifest Signing](#)

Discusses strong-name signing, which can be used to uniquely identify software components, preventing name spoofing.

Securing applications

2/16/2018 • 1 min to read • [Edit Online](#)

While most applications possess common security challenges, each application domain possesses security challenges of its own.

General Security Considerations

Each language has its own security considerations and challenges.

[Security Best Practices](#)

Provides information on security features and practices available when working in Visual C++.

[Security and Programming \(C# and Visual Basic\)](#)

Provides information about the top three security concerns for Visual Basic and C# developers: privileges, Web applications, and Visual Studio setup.

Securing Mobile Applications

As the popularity of mobile devices increases, the security of the information and data on these devices becomes more important.

[Security Considerations for Devices](#)

Describes several factors influencing security policy for devices.

[Security Goals for the .NET Compact Framework](#)

Describes goals for .NET Compact Framework security.

[Designing Secure Mobile Web Form Pages](#)

Discusses planning, implementing, and supporting security in wireless networks and mobile devices.

Securing Web Applications

A poorly written Web page can compromise the integrity and security of an entire server and potentially an entire network. Therefore, you must review security considerations in planning your Web application.

[ASP.NET Security Architecture](#)

Provides an overview of ASP.NET infrastructure and subsystem relationships, as related to security.

[ASP.NET Web Application Security](#)

Details how to address authorization and authentication issues in ASP.NET.

[How to: Use Transport Security](#)

Describes how to use transport security for authentication when you connect to a WCF service.

Securing Desktop Applications

Designing security for desktop applications is an essential step in application development.

[Windows Forms Security](#)

Provides an overview of the Windows Forms security implementation.

See also

- Security

Maintaining Security

3/12/2018 • 1 min to read • [Edit Online](#)

It is often said that the price of security is constant vigilance. Despite your best dedication to security during the design and development of your application, you should assume that security flaws will arise after deployment. By auditing your application and analyzing event logs, you may discover some previously hidden flaws.

In addition, not only must you remain vigilant about your own application, you must also keep current on security threats and flaws for the platform on which your application runs and for other products on which your application depends.

[Security, Privacy, and Accounts](#)—Get help with security, privacy, and user accounts, including info about viruses, passwords, parental controls, firewalls, and drive encryption..

[Microsoft Security Updates Bulletins](#)—This page makes it easy to find previously released bulletins. Intended for IT professionals, security bulletins provide detailed information regarding security updates.

[Microsoft Baseline Security Analyzer](#)—The Microsoft Baseline Security Analyzer (MBSA) is a tool that enables an individual home user, a corporate user, or an administrator to scan one or more Windows-based computers for common security configuration mistakes.

User Permissions and Visual Studio

1/22/2018 • 1 min to read • [Edit Online](#)

For reasons of security you should run Visual Studio as a normal user whenever possible.

WARNING

You should also make sure not to compile, launch, or debug any Visual Studio solution that does not come from a trusted person or a trusted location.

You can do nearly everything in the Visual Studio IDE as a normal user, but, you need administrator permissions to complete the following tasks:

| AREA | TASK | FOR MORE INFORMATION |
|-------------------|---|---|
| Installation | Install Visual Studio. | Install Visual Studio |
| | Installing, updating, or removing local Help content. | Install and Manage Local Content |
| Application types | Developing solutions for SharePoint. | Requirements for Developing SharePoint Solutions |
| | Acquiring a developer license for Microsoft Store. | Get a developer license |
| Toolbox | Adding classic COM controls to Toolbox . | Toolbox |
| Add-ins | Installing and using add-ins that were written by using classic COM in the IDE. | Creating Add-ins and Wizards |
| Building | Using post-build events that register a component. | Understanding Custom Build Steps and Build Events |
| | Including a registration step when you build C++ projects. | Understanding Custom Build Steps and Build Events |
| Debugging | Debugging applications that run with elevated permissions. | Debugger Settings and Preparation |
| | Debugging applications that run under a different user account, such as ASP.NET websites. | Debugging ASP.NET and AJAX Applications |
| | Debugging in Zone for XAML Browser Applications (XBAP). | WPF Host (PresentationHost.exe) |
| | Using the emulator to debug cloud service projects for Microsoft Azure. | Debugging a Cloud Service in Visual Studio |

| AREA | TASK | FOR MORE INFORMATION |
|-------------------|---|--|
| | Configuring a firewall for remote debugging. | Remote Debugging |
| Performance tools | Profiling an application. | Beginners Guide to Performance Profiling |
| Deployment | Deploying a web application to Internet Information Services (IIS) on a local computer. | Deploying an ASP.NET Web Application to a Hosting Provider using Visual Studio or Visual Web Developer: Deploying to IIS as a Test Environment |

Running Visual Studio as an Administrator

You can launch Visual Studio with administrative permissions each time you start the IDE, or you can modify the application shortcut to always run with administrative permissions. For more information, see Windows Help.

To run Visual Studio with administrative permissions

These instructions are for Windows 10. They are similar for other versions of Windows.

1. Open the **Start** menu, and scroll to Visual Studio 2017.
2. From the right-click or context menu of **Visual Studio 2017**, select **More > Run as administrator**.

When Visual Studio starts, **(Administrator)** appears after the product name in the title bar.

See also

[Porting, Migrating, and Upgrading Visual Studio Projects](#)

[Install Visual Studio](#)

Security Bibliography

3/12/2018 • 1 min to read • [Edit Online](#)

Following are selected resources that contain information about developing secure applications and configuring secure environments:

[Microsoft Security](#)—Learn how Microsoft is working to help you keep your applications and systems secure, from the desktop to network-level systems, with links to security resources for IT professionals, developers, and home users.

[Security Resources on MSDN](#)—Guides you to developer-oriented documentation, code samples, technical articles, and other resources for developing secure applications.

Accessibility Features of Visual Studio

12/22/2017 • 1 min to read • [Edit Online](#)

TIP

To learn more about recent accessibility updates, see the [Accessibility improvements in Visual Studio 2017 version 15.3](#) blog post.

In addition to accessibility features and utilities in Windows, the following features make Visual Studio more accessible for people with disabilities:

- Toolbar button and text enlargement
- Text size options in the editors
- Color customization in the editors
- Keyboard shortcut customization
- Auto-completion for methods and parameters

IMPORTANT

The information in this section applies only to users who license Microsoft products in the United States. If you obtained this product outside of the United States, you can use the subsidiary-information card that came with your software package, or you can visit the [Contact Us page for Microsoft Support](#) and click the **Locate Microsoft Offices Worldwide** link (near the bottom of the page). This page provides a list of telephone numbers and addresses for Microsoft support services. You can contact your subsidiary to find out whether the types of products and services that are described in this section are available in your area. Information about accessibility is available in other languages, including Japanese and French.

For more information, see the following topics:

- [How to: Set IDE Accessibility Options](#)
- [How to: Use the Keyboard Exclusively](#)
- [Default Keyboard Shortcuts](#)
- [Accessibility Tips and Tricks](#)
- [How to: Change Fonts and Colors](#)

See Also

[Accessibility Products and Services from Microsoft](#)

How to: Set IDE Accessibility Options

1/26/2018 • 4 min to read • [Edit Online](#)

TIP

To learn more about recent accessibility updates, see the [Accessibility improvements in Visual Studio 2017 version 15.3](#) blog post.

Visual Studio contains features that make it easier for people who have low vision to read and for people who have limited dexterity to write. These features include changing the size and color of text in editors, changing the size of text and buttons on toolbars, and auto-completion for methods and parameters, to name a few.

In addition, Visual Studio supports Dvorak keyboard layouts, which make the most frequently typed characters more accessible. You can also customize the default shortcut keys available with Visual Studio. For more information, see [Identifying and Customizing Keyboard Shortcuts](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Editors, Dialogs, and Tool Windows

By default, dialog boxes and tool windows in Visual Studio use the same font size and colors as the operating system. The color settings for the frame of the IDE, dialog boxes, toolbars, and tool windows are based a color scheme: light or dark. You can change the current color theme in the [General, Environment, Options Dialog Box](#).

You can also display pop-up windows in the Code view of the editor. These windows can prompt you with available members on the current object and the parameters to complete a function or statement. These windows can be helpful if you have difficulty typing. However, they interfere with focus in the code editor, which can be problematic for some users. You can turn off these windows by opening the Options dialog box and clearing **Auto list members** and **Parameter information** in the **Text Editor, All Languages, General** page in the **Options** dialog box.

You can rearrange the windows in the integrated development environment (IDE) to best suit the way you work. You can dock, float, hide, or automatically hide each tool window.

For more information about how to change window layouts, see [Customizing window layouts](#).

Changing the Size of Text

You can change the settings for text-based tool windows, such as the **Command** window, **Immediate** window, and **Output** window, in the **Fonts and Colors** pane of the **Environment** options in the **Tools** dialog box. When **[All Text Tool Windows]** is selected in the **Show settings for** drop-down list, the default setting is listed as **Default** in the **Item foreground** and **Item background** drop-down lists. You can also change the settings for how text is displayed in the editor.

To change the size of text in text-based tool windows and editors

1. From the **Tools** menu, choose **Options**.
2. Choose **Fonts and Colors** on the **Environment** folder.

3. Select an option on the **Show settings for** drop-down menu.

To change the font size for text in an editor, choose **Text Editor**.

To change the font size for text in text-based tool windows, choose **[All Text Tool Windows]**.

To change the font size for ToolTip text in an editor, choose **Editor Tooltip**.

To change the font size for text in statement completion pop-ups, choose **Statement Completion**.

4. From **Display items**, select **Plain Text**.

5. In **Font**, select a new font type.

6. In **Size**, select a new font size.

NOTE

To reset the text size for text-based tool windows and editors, choose **Use Defaults**.

7. Choose **OK**.

Changing the Colors used in the IDE

You can also choose to change the default colors for text, margin indicators, white space, and code elements in the editor.

NOTE

To use high contrast colors for all application windows on your operating system, press Left **ALT+Left SHIFT+PRINT SCREEN**. If Visual Studio is open, close and reopen Visual Studio to fully implement high contrast colors.

To change the color of items in the editor

1. From the **Tools** menu, choose **Options**.

2. Choose **FONTs and Colors** from the **Environment** folder.

3. In **Show settings for**, choose **Text Editor**.

4. From **Display items**, select an item whose display you need to change, such as **Plain Text, Indicator Margin, Visible White Space, HTML Attribute Name, or XML Attribute**.

5. Select display settings from the following options: **Item foreground**, **Item background**, and **Bold**.

6. Choose **OK**.

Toolbars

To improve toolbar usability and accessibility, you can add text to toolbar buttons.

To assign text to toolbar buttons

1. From the **Tools** menu, choose **Customize**.

2. In the **Customize** dialog box, select the **Commands** tab.

3. Select **Toolbar** and then choose the toolbar name that contains the button you intend to display text for.

4. In the list, select the command you intend to change.

5. Choose **Modify Selection**.

6. Choose **Image and Text**.

To modify the button's displayed text

1. Re-select **Modify Selection**.
2. Adjacent to In **Name**, insert provide a new caption for the selected button.

See Also

[Accessibility Features of Visual Studio](#)

[Resources for Designing Accessible Applications](#)

How to: Use the Keyboard Exclusively

12/22/2017 • 2 min to read • [Edit Online](#)

TIP

To learn more about recent accessibility updates, see the [Accessibility improvements in Visual Studio 2017 version 15.3](#) blog post.

Visual Studio provides many default shortcut key combinations to make it easy to navigate and code within the integrated development environment (IDE). For a full listing of shortcut keys used in Visual Studio, see [Default Keyboard Shortcuts](#). For information on keyboard shortcuts available for other Microsoft products, see <http://www.microsoft.com/enable/products/keyboard.aspx>.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Toolbox Controls

You can add a control on the Toolbox to a form or designer using the keyboard.

To add controls from the Toolbox to a designer from the keyboard

1. On the menu bar, choose **View, Toolbox**.
2. Use the Ctrl + Up Arrow or Ctrl + Down Arrow keys to move among the sections in the current Toolbox tab.
3. Use the Up Arrow or **Down Arrow** keys to move among the controls.
4. After the control is selected, use the Enter key.

The control is added to the form or designer.

Dialog Box Options

You can move among the options in a dialog and change option settings using the keyboard.

To set dialog box options from the keyboard

1. Use **TAB** or **SHIFT + TAB** to move up and down through the controls in the dialog box.
2. To change option settings:
 - For radio buttons, use **UP ARROW** and **DOWN ARROW** to change the selection.
 - For check boxes, use **SPACEBAR** to select or unselect.
 - For drop-down lists, use **ALT + DOWNARROW** to display items and then use **UPARROW** and **DOWNARROW** to change the selected item.
 - For buttons, press **ENTER** to invoke.
 - For grids, use the Arrow keys to navigate. For drop-down lists in grids, use **SHIFT + ALT +**

DOWNDARROW to display items and then use **UPARROW** and **DOWNDARROW** to change the selected item.

Window and File Navigation

The IDE provides several ways for you to move among open tool and document windows using the keyboard. You can also move and dock tool windows in different locations using the keyboard.

To navigate among windows and files in the IDE from the keyboard

- To move among files in an editor or designer, choose CTRL + TAB keys to display the IDE Navigator with **Active Files** selected. Choose the Enter key to navigate to the highlighted file.
- To move among docked tool windows, choose Alt + F7 to display the IDE Navigator with **Active Tool Windows** selected. Choose the Enter key to navigate to the highlighted window.

To move and dock tool windows from the keyboard

1. Navigate to the tool window you intend to move and give it focus.
2. On the **Window** menu, select the **Dockable** option.
3. Press **ALT + Space** and then choose **Move**.

The docking guide diamond appears.

4. Use the **ARROW** keys to move the window to a new location.

The mouse pointer moves with the window as you use the **ARROW** keys.

5. When you have reached the new location, use the **ARROW** keys to move the mouse pointer over the correct portion of the guide diamond.

An outline of the tool window appears in the new docking location.

6. Press **ENTER**.

The tool window snaps into place at the new docking location.

See Also

[Identifying and Customizing Keyboard Shortcuts](#)

[Accessibility Tips and Tricks](#)

[Default Keyboard Shortcuts](#)

Accessibility tips and tricks for Visual Studio

12/22/2017 • 3 min to read • [Edit Online](#)

TIP

To learn more about recent accessibility updates, see the [Accessibility improvements in Visual Studio 2017 version 15.3](#) blog post.

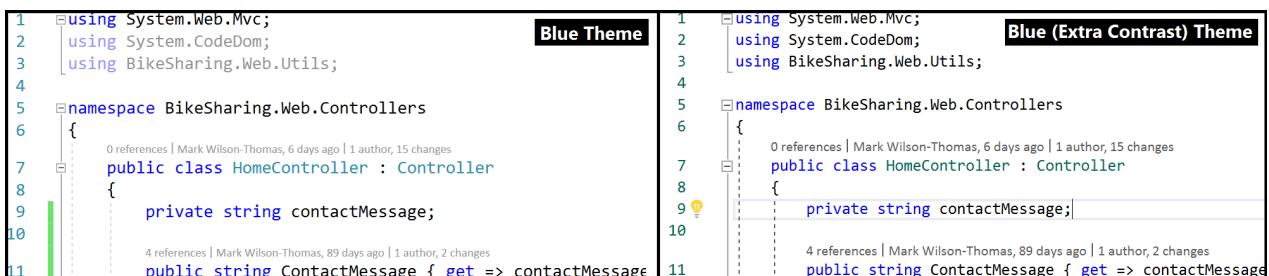
Visual Studio has built-in accessibility features that are compatible with screen readers and other assistive technologies. This topic lists common shortcut key combinations that you can use to perform tasks with the keyboard only, and includes information about using high-contrast themes to improve visibility. As well, it shows you how to use annotations to reveal useful information about your code, and how to set sound cues for build and breakpoint events.

Save your IDE settings

You can customize your IDE experience by saving your window layout, keyboard mapping scheme, and other preferences. For more information, see [Personalize the Visual Studio IDE](#).

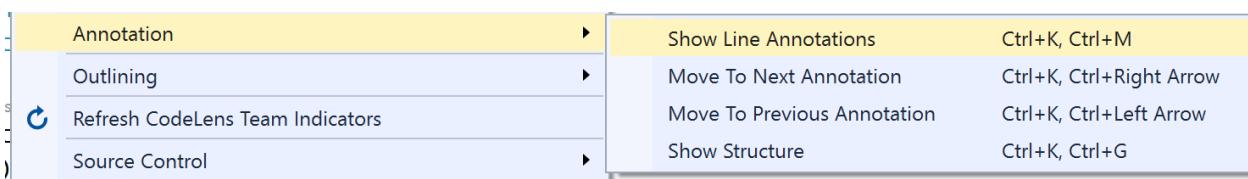
Modify your IDE for high-contrast viewing

For some folks, some colors are more difficult to see. If you want more contrast as you code but do not want to use the typical "High Contrast" themes, we now offer a "Blue (Extra Contrast)" theme.



Use Annotations to reveal useful information about your code

The Visual Studio editor includes many text "adornments" that let you know about characteristics and features at particular points on a line of code, such as lightbulbs, error and warning "squiggles", bookmarks, and so on. You can use the "Show Line Annotations" command set to help you discover and then navigate between these adornments.



Access toolbars by using shortcut key combinations

The Visual Studio IDE has toolbars as do many tool windows. The following shortcut key combinations help you access them.

| FEATURE | DESCRIPTION | KEY COMBINATION |
|----------------------|--|------------------------|
| IDE toolbars | Select the first button on the Standard toolbar. | ALT, CTRL + TAB |
| Tool window toolbars | Move focus to the toolbars in a tool window. NOTE: This works for most tool windows, but only when the focus is in a tool window. Also, you must choose the SHIFT key before the ALT key. In some tool windows, such as Team Explorer, you must hold the SHIFT key for a moment before choosing the ALT key. | SHIFT + ALT |
| Toolbars | Go to the first item in the next toolbar (when a toolbar has focus). | CTRL + TAB |

Other useful shortcut key combinations

Some other useful shortcut key combinations include the following.

| FEATURE | DESCRIPTION | KEY COMBINATION |
|----------------|--|---|
| IDE | Switch High Contrast on and off. NOTE: Standard Windows shortcut | Left ALT + Left SHIFT + PRINT SCREEN |
| Dialog box | Select or clear the check box option in a dialog box. NOTE: Standard Windows shortcut | SPACEBAR |
| Context menus | Open a context (right-click) menu. NOTE: Standard Windows shortcut | SHIFT + F10 |
| Menus | Quickly access a menu item by using its accelerator keys. Choose the ALT key followed by the underlined letters in a menu to activate the command. For example, to view the Open Project dialog box in Visual Studio, you would choose ALT + F + O + P . NOTE: Standard Windows shortcut | ALT + [letter] |
| Toolbox window | Move among Toolbox tabs. and CTRL + UPARROW CTRL + DOWNARROW | |
| Toolbox window | Add a control from the Toolbox to a form or designer. | ENTER |

| FEATURE | DESCRIPTION | KEY COMBINATION |
|---|--|------------------|
| Keyboard, Environment, Options dialog box | Delete a key combination entered in the Press shortcut keys option. | BACKSPACE |

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition.

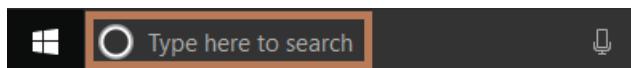
Use the Sound applet to set build and breakpoint cues

You can use the Sound applet in Windows to assign a sound to Visual Studio program events. Specifically, you can assign sounds to the following program events:

- Breakpoint hit
- Build canceled
- Build failed
- Build succeeded

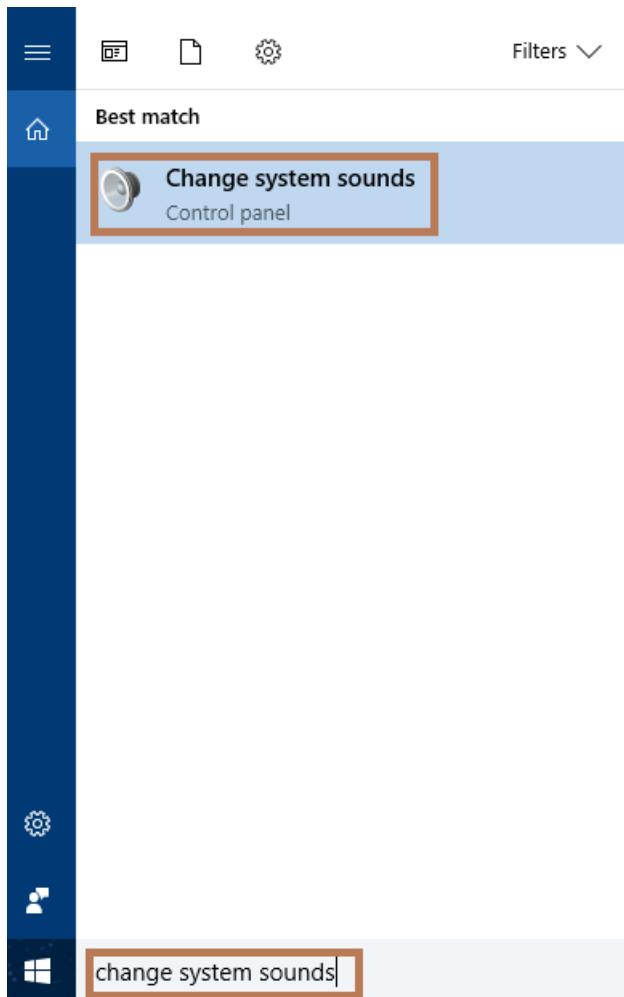
Here's how.

1. In the **Search** box on a computer running Windows 10, type **Change system sounds**.



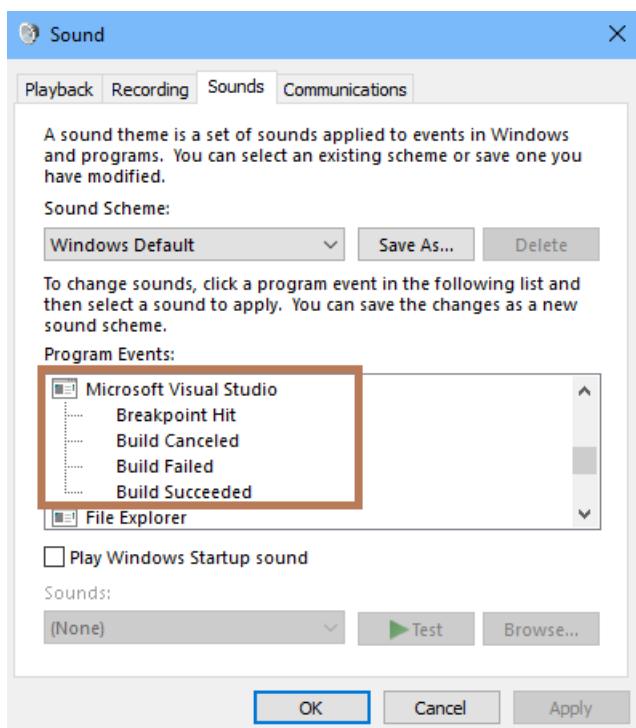
(Alternatively, if you have Cortana enabled, say "Hey Cortana", and then say "Change system sounds".)

2. Double-click **Change system sounds**.



3. In the **Sound** dialog box, click the **Sounds** tab.

Then, in **Program Events**, scroll to **Microsoft Visual Studio**, and select the sounds that you want to apply to the events that you choose.



4. Click **OK**.

See also

- [Accessibility Features of Visual Studio](#)
 - [How to: Customize Menus and Toolbars in Visual Studio](#)
- [Personalize the Visual Studio IDE](#)
- [Microsoft Accessibility](#)

Accessibility Products and Services from Microsoft

12/22/2017 • 3 min to read • [Edit Online](#)

TIP

To learn more about recent accessibility updates to Visual Studio, see the [Accessibility improvements in Visual Studio 2017 version 15.3](#) blog post.

Microsoft is committed to making its products and services easier for everyone to use. The following topics provide information about the features, products, and services that make Microsoft Windows more accessible for people with disabilities:

- [Accessibility Features of Windows](#)
- [Documentation in Alternative Formats](#)
- [Customer Service for People with Hearing Impairments](#)
- [For More Information](#)

NOTE

The information in this section may apply only to users who license Microsoft products in the United States. If you obtained this product outside of the United States, you can use the subsidiary information card that came with your software package or visit the [Microsoft Accessibility Web site](#) for a list of Microsoft support services telephone numbers and addresses. You can contact your subsidiary to find out whether the type of products and services described in this section are available in your area. Information about accessibility is available in other languages, including Japanese and French.

Accessibility Features of Windows

The Windows operating system has many built-in accessibility features that are useful for individuals who have difficulty typing or using a mouse, are blind or have low vision, or who are deaf or hard-of-hearing. The features are installed during Setup. For more information about these features, see Help in Windows and the [Microsoft Accessibility Web site](#).

Free Step-by-Step Tutorials

Microsoft offers a series of step-by-step tutorials that provide detailed procedures for adjusting the accessibility options and settings on your computer. This information is presented in a side-by-side format so that you can learn how to use the mouse, the keyboard, or a combination of both.

To find step-by-step tutorials for Microsoft products, see the [Microsoft Accessibility Web site](#).

Assistive Technology Products for Windows

A wide variety of assistive technology products are available to make computers easier to use for people with disabilities. You can search a catalog of assistive technology products that run on Windows at the [Microsoft Accessibility Web site](#).

If you use assistive technology, be sure to contact your assistive technology vendor before you upgrade your software or hardware to check for possible compatibility issues.

Documentation in Alternative Formats

If you have difficulty reading or handling printed materials, you can obtain the documentation for many Microsoft products in more accessible formats. You can view an index of accessible product documentation on the [Microsoft Accessibility Web site](#).

In addition, you can obtain additional Microsoft publications from Recording for the Blind & Dyslexic, Inc (RFB&D). RFB&D distributes these documents to registered, eligible members of their distribution service. For information about the availability of Microsoft product documentation and books from Microsoft Press, contact:

RECORDING FOR THE BLIND & DYSLEXIC, INC.

20 Roszel Road

Princeton, NJ 08540

Telephone number from within the United States: (800) 221-4792

Telephone number from outside the United States and Canada: (609) 452-0606

Fax: (609) 987-8116

Recording for the Blind & Dyslexic website: <http://www.rfbd.org>.

Web addresses can change, so you might be unable to connect to the website or sites mentioned here.

Customer Service for People with Hearing Impairments

If you are deaf or hard-of-hearing, complete access to Microsoft product and customer services is available through a text telephone (TTY/TDD) service:

- For customer service, contact Microsoft Sales Information Center at (800) 892-5234 between 6:30 AM and 5:30 PM Pacific Time, Monday through Friday, excluding holidays.
- For technical assistance in the United States, contact Microsoft Product Support Services at (800) 892-5234 between 6:00 AM and 6:00 PM Pacific Time, Monday through Friday, excluding holidays. In Canada, dial (905) 568-9641 between 8:00 AM and 8:00 PM Eastern Time, Monday through Friday, excluding holidays.

Microsoft Support Services are subject to the prices, terms, and conditions in place at the time the service is used.

For More Information

For more information about how accessible technology for computers helps to improve the lives of people with disabilities, see the [Microsoft Accessibility Web site](#).

See Also

[Resources for Designing Accessible Applications](#)

[Accessibility Features of Visual Studio](#)

Resources for Designing Accessible Applications

1/26/2018 • 1 min to read • [Edit Online](#)

Use the following links to find information about technologies that support accessible design as well as tips and examples for developing accessible Windows applications and Web sites. General information on accessibility can be found online at <http://www.microsoft.com/enable/>.

Technologies

- **Microsoft Active Accessibility** A COM-based technology that improves the way accessibility aids work with applications running on Microsoft Windows. It provides dynamic-link libraries that are incorporated into the operating system as well as a COM interface and application programming elements that provide reliable methods for exposing information about user interface elements. For more information, see [http://msdn.microsoft.com/library/windows/desktop/dd373592\(v=vs.85\).aspx](http://msdn.microsoft.com/library/windows/desktop/dd373592(v=vs.85).aspx).
- **Microsoft .NET Speech Technologies** The Microsoft .NET Speech SDK is a set of Microsoft ASP.NET controls, a Microsoft Internet Explorer Speech add-in, sample applications and documentation that allows Web developers to create, debug and deploy speech-enabled ASP.NET applications. The tools are integrated seamlessly into Microsoft Visual Studio, allowing developers to leverage the familiar development environment. For more information, see <http://msdn.microsoft.com/library/ms950383.aspx>.
- **Understanding SAMI 1.0** Microsoft Synchronized Accessible Media Interchange (SAMI) technology provides a way for developers to caption audio content for PC multimedia. For more information, see [Understanding SAMI 1.0](#).

Windows Applications

- [Walkthrough: Creating an Accessible Windows-based Application](#) This topic provides step-by-step instructions for including the five accessibility requirements for the Certified for Windows logo in a sample Windows application.
- **Guidelines for Keyboard User Interface Design** This technical article describes how to design a Windows application user interface that users can navigate from the keyboard. For more information, see <http://msdn2.microsoft.com/library/ms971323.aspx>.
- **Console Accessibility** This technical article describes the APIs and events used to expose the console in Windows XP for accessibility aids. For more information, see <http://msdn2.microsoft.com/library/ms971319.aspx>.

Web Sites

- [Walkthrough: Accessibility Guidelines for Using Image Controls, Menu Controls, and AutoPostBack](#) This topic provides step-by-step instructions for including accessible controls in a sample Web page as well as some accessibility design tips for the Web.
- **Creating Accessible Web Pages with DHTML** This technical article lists HTML 4.0 elements that are accessible as well as accessible Web design tips. For more information, see <http://msdn2.microsoft.com/library/ms528445.aspx>.

Third-party Resources

- **Web Accessibility Initiative of the World Wide Web Consortium (W3C)** This Web site provides guidelines

and techniques for accessible Web site development. For more information, see <http://www.w3.org/WAI/GL/>.

See Also

[Accessibility Features of Visual Studio](#)

Globalizing and Localizing Applications

1/26/2018 • 1 min to read • [Edit Online](#)

If you plan on distributing your application to an international audience, you'll need to keep several things in mind during the design and development phases. Even if you don't have such plans, a small effort up front can make things considerably easier should your plans change in future versions of your application. Services built into the .NET Framework make it easy to develop a single application that can adapt to different locales using managed development with Visual Studio.

Visual Studio was designed from the start to make developing for an international audience easy by taking advantage of services built into the .NET Framework. The following pages will help introduce you to the internationalization features built into Visual Studio.

In This Section

[Introduction to International Applications Based on the .NET Framework](#)

Introduces the concepts related to developing software for an international market using Visual Studio and the .NET Framework.

[Localizing Applications](#)

Provides links to pages about customizing applications for a given culture.

[Globalizing Applications](#)

Provides links to pages about creating applications that support multiple cultures.

Related Sections

[Best Practices for Developing World-Ready Applications](#)

Provides background information on programming for an international audience.

[Class Library Overview](#)

Introduces the classes, interfaces, and value types that expedite and optimize the development process and provide access to system functionality.

[System.Globalization](#)

Points out the classes in this namespace, which define culture-related information, including the language, the country/region, the calendars in use, the format patterns for dates, currency and numbers, and the sort order for strings.

[System.Resources](#)

Points out the classes and interfaces in this namespace, which allows developers to create, store, and manage various culture-specific resources used in an application.

Introduction to International Applications Based on the .NET Framework

1/26/2018 • 2 min to read • [Edit Online](#)

In Visual Studio, there are two parts to creating a world-ready application: globalization, the process of designing applications that can adapt to different cultures, and localization, the process of translating resources for a specific culture. For general information on designing applications for an international audience, see [Best Practices for Developing World-Ready Applications](#).

The .NET Framework localization model consists of a main assembly that contains both the application code and the fallback resources — strings, images, and other objects for the language in which the application is originally developed. Each localized application will have satellite assemblies, or assemblies which contain only the localized resources. Because the main assembly always contains the fallback resources, if a resource is not found in the localized satellite assembly, the [ResourceManager](#) will attempt to load it in a hierarchical manner, eventually falling back to the resource in the main assembly. The resource fallback system is explained in greater detail in [Hierarchical Organization of Resources for Localization](#).

One localization resource you should consider using is the glossary for all Microsoft localized products. This CSV file contains over 12,000 English terms plus the translations of the terms in up to 59 different languages. The glossary is available for download on the [Microsoft Terminology Translations](#) Web page.

The project system for Windows Forms applications can generate resource files for both the fallback and each desired additional UI culture. The fallback resource file is built into the main assembly, and the culture-specific resource files are then built into satellite assemblies, one for each UI culture. When you build a project, the resource files are compiled from the Visual Studio XML format (.resx) to an intermediate binary format (.resources), which are then embedded in satellite assemblies.

The project system for both Windows Forms and Web Forms allows you to build resource files using an Assembly Resource File template, access the resources, and build your project. Satellite assemblies will be created along with the main assembly.

When a localized application executes, its appearance is determined by two culture values. (A *culture* is a set of user preference information related to the user's language, environment, and cultural conventions.) The UI culture setting determines which resources will be loaded. The UI culture is set as `uiCulture` in Web.config files and page directives, and [CurrentUICulture](#) in Visual Basic or C# code. The culture setting determines formatting of values such as dates, numbers, currency, and so on. The culture is set as `culture` in Web.config files and page directives, [CurrentCulture](#) in Visual Basic or C# code.

See Also

[System.Globalization](#)

[System.Resources](#)

[Globalizing and Localizing Applications](#)

[Security and Localized Satellite Assemblies](#)

Localizing Applications

1/26/2018 • 1 min to read • [Edit Online](#)

Localization is the process of customizing your application for a given culture or locale. Localization consists primarily of translating the user interface.

In This Section

[Hierarchical Organization of Resources for Localization](#)

Explains how localized resources are stored and accessed in Visual Studio.

[Security and Localized Satellite Assemblies](#)

Discusses signing satellite assemblies with public-private key pairs.

[Version Numbers for Main and Localized Satellite Assemblies](#)

Introduces the [SatelliteContractVersionAttribute](#) class, which determines which satellite assemblies work with an application's main assembly.

[Neutral Resources Languages for Localization](#)

Introduces the [NeutralResourcesLanguageAttribute](#) class, which specifies the culture of the resources included in an application's main assembly.

Related Sections

[Introduction to International Applications Based on the .NET Framework](#)

Discusses the concepts related to developing software for an international market using Visual Basic or C#.

[Globalizing Windows Forms](#)

Provides links to pages about creating Windows applications that support multiple cultures.

[Globalization and Localization](#)

Provides links to pages about creating Web applications that support multiple cultures.

[Best Practices for Developing World-Ready Applications](#)

Provides information on programming for an international audience, such as design issues and terminology.

Hierarchical Organization of Resources for Localization

1/26/2018 • 1 min to read • [Edit Online](#)

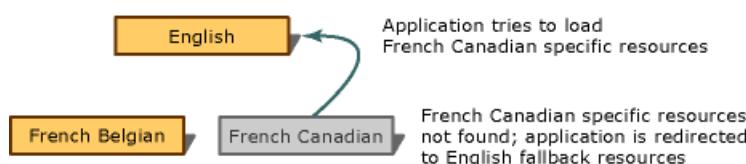
In Visual Studio, localized resources (data such as strings and images appropriate to each culture) are stored in separate files and loaded according to the UI culture setting. To understand how localized resources are loaded, it is useful to think of them as organized in a hierarchical manner.

Kinds of Resources in the Hierarchy

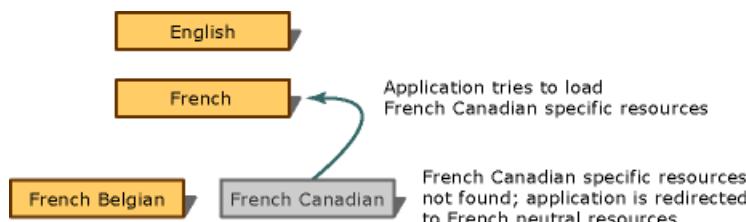
- At the top of the hierarchy sit the fallback resources for your default culture, for example English ("en"). These are the only resources that do not have their own file; they are stored in the main assembly.
- Below the fallback resources are the resources for any neutral cultures. A neutral culture is associated with a language but not a country/region. For example, French ("fr") is a neutral culture. (Note that the fallback resources are also for a neutral culture, but a special one.)
- Below those are the resources for any specific cultures. A specific culture is associated with a language and a country/region. For example, French Canadian ("fr-CA") is a specific culture.

If an application tries to load any localized resource, such as a string, and does not find it, it will travel up the hierarchy until it finds a resource file containing the requested resource.

The best way to store your resources is to generalize them as much as possible. That means to store localized strings, images, and so forth in resource files for neutral cultures rather than specific cultures whenever possible. For instance, if you have resources for the French Belgian ("fr-BE") culture and the resources immediately above are the fallback resources in English, a problem may result when someone uses your application on a system configured for the French Canadian culture. The system will look for a satellite assembly for "fr-CA", not find it, and load the main assembly containing the fallback resource, which is English, instead of loading the French resources. The following picture shows this undesirable scenario.



If you follow the recommended practice of placing as many resources as possible in a neutral resource file for the "fr" culture, the French Canadian user would not see resources marked for the "fr-BE" culture, but he or she would be shown strings in French. The following situation shows this preferred scenario.



See also

[Neutral Resources Languages for Localization](#)
[Security and Localized Satellite Assemblies](#)

[Localizing Applications](#)

[Globalizing and Localizing Applications](#)

Security and Localized Satellite Assemblies

2/16/2018 • 1 min to read • [Edit Online](#)

If your main assembly uses strong naming, satellite assemblies must be signed with the same private key as the main assembly. If the public/private key pair does not match between the main and satellite assemblies, your resources will not be loaded. For more information on signing assemblies, see [How to: Sign an Assembly with a Strong Name](#).

In general, you may need to have your organization's signing group or an external signing organization sign with the private key. This is due to the sensitive nature of the private key: access is often restricted to a few individuals. You can use delayed signing during development. For more information, see [Delay Signing an Assembly](#).

See also

- [Assembly Security Considerations - Key Security Concepts](#)
- [Introduction to International Applications Based on the .NET Framework](#)
- [Localizing Applications](#)
- [Globalizing and Localizing Applications](#)

Version Numbers for Main and Localized Satellite Assemblies

12/22/2017 • 1 min to read • [Edit Online](#)

The [SatelliteContractVersionAttribute](#) class provides versioning support for a main assembly that uses localized resources by means of the resource manager. Applying the [SatelliteContractVersionAttribute](#) to an application's main assembly allows you to update and re-deploy the assembly without updating its satellite assemblies. For example, you can use the [SatelliteContractVersionAttribute](#) class with a service pack that doesn't introduce new resources without rebuilding and redeploying your satellite assemblies. For your localized resources to be available, the satellite contract version of your main assembly must match the [AssemblyVersionAttribute](#) class of your satellite assemblies. You must specify an exact version number in the [SatelliteContractVersionAttribute](#); wildcard characters such as "*" are not allowed. For more information, see [Retrieving Resources](#).

Updating Assemblies

The [SatelliteContractVersionAttribute](#) class allows you to update a main assembly without having to update your satellite assembly, or vice versa. When the main assembly is updated, its assembly version number is changed. If you want to continue using the existing satellite assemblies, change the main assembly's version number but leave the satellite contract version number the same. For example, in your first release your main assembly version may be 1.0.0.0. The satellite contract version and the assembly version of the satellite assembly will also be 1.0.0.0. If you need to update your main assembly for a service pack, you can change the assembly version to 1.0.0.1, while keeping the satellite contract version and the satellite's assembly version as 1.0.0.0.

If you need to update a satellite assembly but not your main assembly, you change the [AssemblyVersionAttribute](#) of the satellite assembly. Along with your satellite assembly, you will have to ship a policy assembly that states that your new satellite assembly is compatible with your old satellite assembly. For more information on policies, see [How the Runtime Locates Assemblies](#).

The following code shows how to set the satellite contract version. The code can be placed in either a build script or in the AssemblyInfo.vb or AssemblyInfo.cs file.

```
<Assembly: SatelliteContractVersionAttribute("4.3.2.1")>
```

```
[assembly: SatelliteContractVersionAttribute("4.3.2.1")]
```

See Also

[How the Runtime Locates Assemblies](#)

[Setting Assembly Attributes](#)

[Security and Localized Satellite Assemblies](#)

[Localizing Applications](#)

[Globalizing and Localizing Applications](#)

Neutral Resources Languages for Localization

12/22/2017 • 1 min to read • [Edit Online](#)

The [NeutralResourcesLanguageAttribute](#) class specifies the culture of the resources included in the main assembly. This attribute is used as a performance enhancement, so that the [ResourceManager](#) object does not search for resources that are included in the main assembly.

The following code shows how to set the neutral resources language. The code can be placed in either a build script or in the AssemblyInfo.vb or AssemblyInfo.cs file.

```
' Set neutral resources language for assembly.  
<Assembly: NeutralResourcesLanguageAttribute("en")>
```

```
// Set neutral resources language for assembly.  
[assembly: NeutralResourcesLanguageAttribute("en")]
```

See Also

[ResourceManager](#)

[Introduction to International Applications Based on the .NET Framework](#)

[Hierarchical Organization of Resources for Localization](#)

[Localizing Applications](#)

[Globalizing and Localizing Applications](#)

Globalizing Applications

1/26/2018 • 1 min to read • [Edit Online](#)

Globalization is the process of designing and developing a software product that functions for multiple cultures. This section applies to both Windows Forms and Web Forms pages.

In This Section

[Culture-Specific Classes for Global Windows Forms and Web Forms](#)

Discusses classes that format dates, time, numbers, currency, and other information according to the culture setting.

Related Sections

[Globalizing Windows Forms](#)

Provides links to Help pages about globalization of Windows Forms.

[Globalization and Localization](#)

Provides links to Help pages about globalization of Web Forms pages.

[Introduction to International Applications Based on the .NET Framework](#)

Discusses the essential concepts about developing software for an international market using Visual Basic or C#.

[Best Practices for Developing World-Ready Applications](#)

Provides information on programming for an international audience, such as design issues and terminology.

Culture-Specific Classes for Global Windows Forms and Web Forms

1/26/2018 • 1 min to read • [Edit Online](#)

Each culture has different conventions for displaying dates, time, numbers, currency, and other information. The [System.Globalization](#) namespace contains classes that can be used to modify how culture-specific values are displayed, such as [DateTimeFormatInfo](#), [Calendar](#), and [NumberFormatInfo](#).

Using the Culture Setting

Use the culture setting, stored either in the application or in the [Regional Options](#) control panel, to automatically determine the culture conventions at run time and format the information accordingly. For more information on setting the culture, see [How to: Set the Culture and UI Culture for ASP.NET Web Page Globalization](#). Classes that automatically format information according to the culture setting are called culture-specific. Some culture-specific methods are [System.IFormattable.ToString](#), [System.Console.WriteLine](#), and [System.String.Format](#). Some culture-specific functions (in the Visual Basic language) are `MonthName` and `WeekDayName`.

For example, the following code shows how you can use the [ToString](#) method to format currency for the current culture:

```
' Put the Imports statements at the beginning of the code module
Imports System.Threading
Imports System.Globalization
' Display a number with the culture-specific currency formatting
Dim MyInt As Integer = 100
Console.WriteLine(MyInt.ToString("C", Thread.CurrentThread.CurrentCulture))
```

```
// Put the using statements at the beginning of the code module
using System.Threading;
using System.Globalization;
// Display a number with the culture-specific currency formatting
int myInt = 100;
Console.WriteLine(myInt.ToString("C", Thread.CurrentThread.CurrentCulture));
```

If the culture is set to "fr-FR", you will see this in the output window:

```
100,00
```

If the culture is set to "en-US", you will see this in the output window:

```
$100.00
```

See also

[System.IFormattable.ToString](#)

[DateTimeFormatInfo](#)

[NumberFormatInfo](#)

[Calendar](#)

[System.Console.WriteLine](#)

[System.String.Format](#)

[Globalizing and Localizing Applications](#)

Creating Applications in Bi-directional Languages

1/26/2018 • 4 min to read • [Edit Online](#)

You can use Visual Studio to create applications that correctly display text in languages written right-to-left, including Arabic and Hebrew. For some features, you can simply set properties. In other cases, you must implement features in code.

NOTE

In order to enter and display bi-directional languages, you must be working with a version of Windows that is configured with the appropriate language. This can either be an English version of Windows with the appropriate language pack installed, or the appropriately localized version of Windows.

Types of Application that Support Bi-Directional Languages

1. Windows applications. You can create fully bi-directional applications that include support for bi-directional text, right-to-left reading order, and mirroring (reversing the layout of windows, menus, dialog boxes, and so on). Except for mirroring, these features are available by default or as property settings. Mirroring is supported inherently for some features, such as message boxes. However, in other cases you must implement mirroring in code. For more information, see [Bi-Directional Support for Windows Forms Applications](#).
2. Web applications. Web services support and receiving sending UTF-8 and Unicode text, making them suitable for applications involving bi-directional languages. Web client applications rely on browsers for their user interface, so the degree of bi-directional support in a Web application is dependent on how well the user's browser supports those bi-directional features. In Visual Studio, you can create applications with support for Arabic or Hebrew text, right-to-left reading order, file encoding, and local culture settings. For more information, see [Bidirectional Support for ASP.NET Web Applications](#).
3. Console applications. Console applications do not include text support for bi-directional languages. This is a consequence of how Windows works with console applications.

Visual Studio Features That Are Fully Supported

At design time in Visual Studio, you can use bi-directional languages in these ways:

- **Text entry** Visual Studio supports Unicode, so if your system is set to the appropriate locale and input language, you can enter text in Arabic or Hebrew. (Arabic support includes Kashida and Diacritics.)
- **Object names** You can use bi-directional languages to assign names to solutions, projects, files, folders, and so on. In code, you can use bi-directional languages for the names of variables, classes, object, attributes, metadata, and other elements.
- **File encoding** You can save and open files with a language-specific or Unicode encoding. For more information, see [How to: Save and Open Files with Encoding](#).

Features with Limited or No Support

Other features common to bi-directional language applications are not fully supported in Visual Studio, or in some cases, not at all. These include:

- **Right-to-left reading order** By default, text-entry controls you use in Visual Studio use left-to-right reading order. In most cases, you can use standard Windows gestures to switch reading order. For example, you can press Ctrl+Right Shift to switch the Properties window to support right-to-left reading order for property values.

However, right-to-left reading order is not supported everywhere in Visual Studio. Exceptions include:

- Check boxes, drop-down lists, and other controls in Visual Studio dialog boxes always use left-to-right reading order.
- The code editor (and text editor) does not support right-to-left reading order. You can enter text in a bi-directional language, but the reading order is always left-to-right.

Naming Things Using Arabic or Hebrew Text

You can use Arabic or Hebrew text to assign names to folders, variables, or other objects. When working with Arabic, you can use any Arabic characters including Kashida and Diacritics.

The following elements can be named using Arabic or Hebrew and will be handled correctly in Visual Studio:

- Solution, project, and file names, including any folders you include in the project path. Solution Explorer will display solution and element names correctly.
- File contents. You can open or save files with Unicode encoding or with a selected code page.

NOTE

The code editor is a special case. For details, see below.

- Data elements. **Server Explorer** will display these elements correctly and allow you to edit them.
- Elements copied to the Windows Clipboard.
- Attributes and metadata.
- Property values. You can use Arabic or Hebrew text in the Properties window. The window allows you to switch between right-to-left and left-to-right reading order using standard Windows keystrokes (CTRL+RightShift for right-to-left, and CTRL+LeftShift for left-to-right).
- Code and literal text. In the code editor (which is also the text editor), you can use Arabic or Hebrew to name classes, functions, variables, properties, string literals, attributes, and so on. However, the editor does not support right-to-left reading order; text always starts at the left margin.

TIP

It is recommended that you place string literals in resource files instead of hard-coding them into your programs. For more information, see [Resources in desktop apps \(.NET Framework\)](#).

NOTE

You must be consistent in how you refer to objects named in these languages. For example, if you use Kashida in naming an Arabic variable, you must always use Kashida when referring to that variable, or errors will result.

- Code comments. You can create comments in Arabic or Hebrew. You can also use these languages in the comment builder tool.

See also

[Bi-Directional Support for Windows Forms Applications](#)

[Bidirectional Support for ASP.NET Web Applications](#)

[Globalizing Applications](#)

[Localizing Applications](#)

Template parameters

1/5/2018 • 2 min to read • [Edit Online](#)

By using parameters in your templates, you can replace the values of key portions of the template, such as class names and namespaces, when the template is instantiated. These parameters are replaced by the template wizard that runs in the background when a user chooses **OK** or **Add** in the **New Project** or **Add New Item** dialog boxes.

Declaring and enabling template parameters

Template parameters are declared in the format `$parameter$`. For example:

- `$safeprojectname$`
- `$guid1$`
- `$guid5$`

To enable parameter substitution in templates

1. In the .vstemplate file of the template, locate the `ProjectItem` element that corresponds to the item for which you want to enable parameter replacement.
2. Set the `ReplaceParameters` attribute of the `ProjectItem` element to `true`.
3. In the code file for the project item, include parameters where appropriate. For example, the following parameter specifies that the safe project name be used for the namespace in a file:

```
namespace $safeprojectname$
```

Reserved template parameters

The following table lists the reserved template parameters that can be used by any template.

| PARAMETER | DESCRIPTION |
|-------------|--|
| clrversion | Current version of the common language runtime (CLR). |
| guid[1-10] | A GUID used to replace the project GUID in a project file. You can specify up to 10 unique GUIDs (for example, <code>guid1</code>). |
| itemname | The name provided by the user in the Add New Item dialog box. |
| machinename | The current computer name (for example, Computer01). |
| projectname | The name provided by the user in the New Project dialog box. |

| PARAMETER | DESCRIPTION |
|------------------------|--|
| registeredorganization | The registry key value from HKLM\Software\Microsoft\Windows NT\CurrentVersion\RegisteredOrganization. |
| rootnamespace | The root namespace of the current project. This parameter applies only to item templates. |
| safeitemname | The name provided by the user in the Add New Item dialog box, with all unsafe characters and spaces removed. |
| safeprojectname | The name provided by the user in the New Project dialog box, with all unsafe characters and spaces removed. |
| time | The current time in the format DD/MM/YYYY 00:00:00. |
| SpecificSolutionName | The name of the solution. When "create solution directory" is checked, <code>SpecificSolutionName</code> has the solution name. When "create solution directory" is not checked, <code>SpecificSolutionName</code> is blank. |
| userdomain | The current user domain. |
| username | The current user name. |
| webnamespace | The name of the current Web site. This parameter is used in the Web form template to guarantee unique class names. If the Web site is at the root directory of the Web server, this template parameter resolves to the root directory of the Web Server. |
| year | The current year in the format YYYY. |

NOTE

Template parameters are case-sensitive.

Custom template parameters

You can specify your own template parameters and values, in addition to the default reserved template parameters that are used during parameter replacement. For more information, see [CustomParameters element \(Visual Studio templates\)](#).

Example: Using the project name for a file name

You can specify variable file names for project items by using a parameter in the `TargetFileName` attribute.

The following example specifies that an executable file's name uses the project name, specified by `$projectname$`.

```
<TemplateContent>
  <ProjectItem
    ReplaceParameters="true"
    TargetFileName="$projectname$.exe">
    File1.exe
  </ProjectItem>
  ...
</TemplateContent>
```

Example: Using the safe project name for the namespace name

To use the safe project name for the namespace in a C# class file, use the following syntax:

```
namespace $safeprojectname$
{
  public class Class1
  {
    public Class1()
    { }
  }
}
```

In the .vstemplate file for the project template, include the `ReplaceParameters="true"` attribute when you reference the file:

```
<TemplateContent>
  <ProjectItem ReplaceParameters="true">
    Class1.cs
  </ProjectItem>
  ...
</TemplateContent>
```

See also

[Customizing templates](#)

[How to: Create project templates](#)

Visual Studio Template Schema Reference

12/22/2017 • 1 min to read • [Edit Online](#)

This section contains information about XML elements in .vstemplate files, which are files that store metadata for project templates, item templates, and Starter Kits.

You can use vstemplate.xsd to validate custom .vstemplate files. This file is available at ..\Visual Studio installation folder\Xml\Schemas\1033\vstemplate.xsd.

| ELEMENT | CHILD ELEMENTS | ATTRIBUTES |
|-----------------------------|-----------------------|---------------|
| AppliesTo | None | None |
| Assembly (Template) | -- | -- |
| Assembly (Wizard Extension) | -- | -- |
| BuildProjectOnload | -- | -- |
| CreateInPlace | -- | -- |
| CreateNewFolder | -- | -- |
| CustomDataSignature | -- | -- |
| CustomParameter | -- | Name Value |
| CustomParameters | CustomParameter | -- |
| DefaultName | -- | -- |
| Description | -- | Package ID |
| EnableEditOfLocationField | -- | -- |
| EnableLocationBrowseButton | -- | -- |
| Folder | ProjectItem Folder | Name |
| | [deprecated] | -- |
| FullClassName | -- | -- |
| Hidden | -- | -- |

| ELEMENT | CHILD ELEMENTS | ATTRIBUTES |
|----------------------------------|---------------------------------------|---|
| Icon | -- | Package ID |
| LocationField | -- | -- |
| LocationFieldMRUPrefix | -- | -- |
| MaxFrameworkVersion | -- | -- |
| Name | -- | Package ID |
| NumberOfParentCategoriesToRollUp | -- | -- |
| PreviewImage | -- | -- |
| Project | Folder ProjectItem | File TargetFileName ReplaceParameters |
| ProjectCollection | ProjectTemplateLink SolutionFolder | -- |
| ProjectItem (Item Templates) | -- | SubType CustomTool ItemType ReplaceParameters TargetFileName |
| ProjectItem (Project Templates) | -- | TargetFileName ReplaceParameters OpenInEditor OpenOrder OpenInWebBrowser OpenInHelpBrowser |
| ProjectSubType | -- | -- |
| ProjectTemplateLink | -- | ProjectName |
| ProjectType | -- | -- |

| ELEMENT | CHILD ELEMENTS | ATTRIBUTES |
|--------------------------|---|-------------|
| PromptForSaveOnCreation | -- | -- |
| ProvideDefaultName | -- | -- |
| Reference | Assembly | -- |
| References | Reference | -- |
| RequiredFrameworkVersion | -- | -- |
| RequiredPlatformVersion | -- | Version |
| SDKReference | -- | Package |
| ShowByDefault | -- | -- |
| SolutionFolder | ProjectTemplateLink SolutionFolder | Name |
| SortOrder | -- | -- |
| SupportsCodeSeparation | -- | -- |
| SupportsLanguageDropDown | -- | -- |
| SupportsMasterPage | -- | -- |
| TargetPlatformName | RequiredPlatformVersion | -- |
| TemplateContent | ProjectCollection Project References ProjectItem CustomParameters | BuildOnLoad |

| ELEMENT | CHILD ELEMENTS | ATTRIBUTES |
|--------------|---|------------|
| TemplateData | Name Description Icon PreviewImage ProjectType ProjectSubType TemplateID TemplateGroupID SortOrder CreateNewFolder DefaultName ProvideDefaultName PromptForSaveOnCreation EnableLocationBrowseButton EnableEditOfLocationField Hidden DisplayInParentCategories LocationFieldMRUPrefix NumberOfParentCategoriesToRollUp CreateInPlace BuildOnLoad BuildProjectOnload ShowByDefault LocationField SupportsMasterPage SupportsCodeSeparation SupportsLanguageDropDown RequiredFrameworkVersion FrameworkVersion MaxFrameworkVersion CustomDataSignature | -- |

| ELEMENT | targetPlatformName CHILD ELEMENTS | ATTRIBUTES |
|-----------------|--|-----------------|
| TemplateGroupID | -- | -- |
| TemplateID | -- | -- |
| VSTemplate | TemplateData TemplateContent WizardExtension WizardData | Type Version |
| WizardData | -- | Name |
| WizardExtension | Assembly FullClassName | -- |

See Also

[Creating Project and Item Templates](#)

[How to: Create Starter Kits](#)

View call hierarchy

1/13/2018 • 3 min to read • [Edit Online](#)

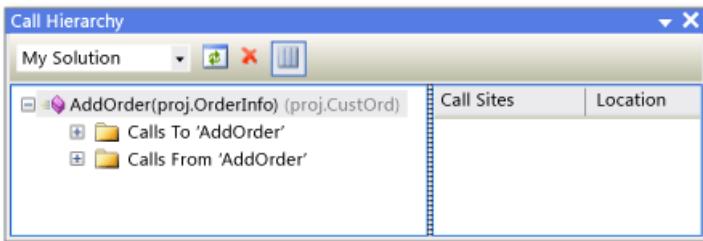
By viewing the call hierarchy for your code, you can navigate all calls to and from a selected method, property, or constructor. This enables you to better understand how code flows, and to evaluate the effects of changes to code. You can examine several levels of code to view complex chains of method calls and additional entry points to the code. This enables you to explore all possible execution paths.

In Visual Studio, you can view a call hierarchy at design time. This means you don't have to set a breakpoint and start the debugger to view the run time call stack.

Use the Call Hierarchy window

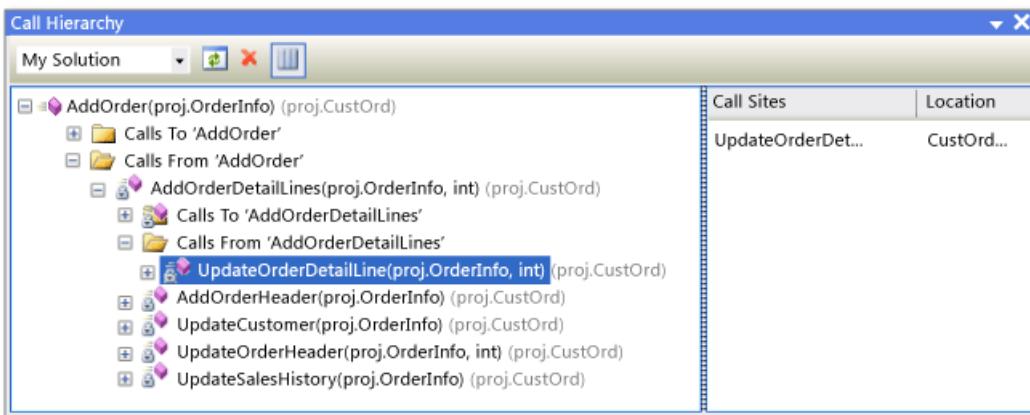
To display the **Call Hierarchy** window, right-click the name of a method, property, or constructor call, and then click **View Call Hierarchy**.

The member name appears in a tree view pane in the **Call Hierarchy** window. If you expand the member node, **Calls To member name** and **Calls From member name** subnodes appear. The following illustration shows these nodes in the **Call Hierarchy** window.



- If you expand the **Calls To** node, all members that call the selected member are displayed.
- If you expand the **Calls From** node, all members that are called by the selected member are displayed.

You can then expand each of these subnode members into **Calls To** and **Calls From** nodes. This enables you to navigate into the stack of callers, as shown in the following illustration.



For members that are defined as either virtual or abstract, an **Overrides method name** node appears. For interface members, an **Implements method name** node appears. These expandable nodes appear at the same level as the **Calls To** and **Calls From** nodes.

The **Search Scope** box on the toolbar contains choices for **My Solution**, **Current Project**, and **Current Document**.

When you select a child member in the **Call Hierarchy** tree view pane:

- The **Call Hierarchy** details pane displays all lines of code in which that child member is called from the parent member.
- The **Code Definition Window**, if open, displays the code for the selected member (C++ only). For more information about this window, see [Viewing the Structure of Code](#).

NOTE

The Call Hierarchy feature does not find method group references, which includes places where a method is added as an event handler or is assigned to a delegate. To find all references to a method, you can use the **Find All References** command.

Shortcut menu items

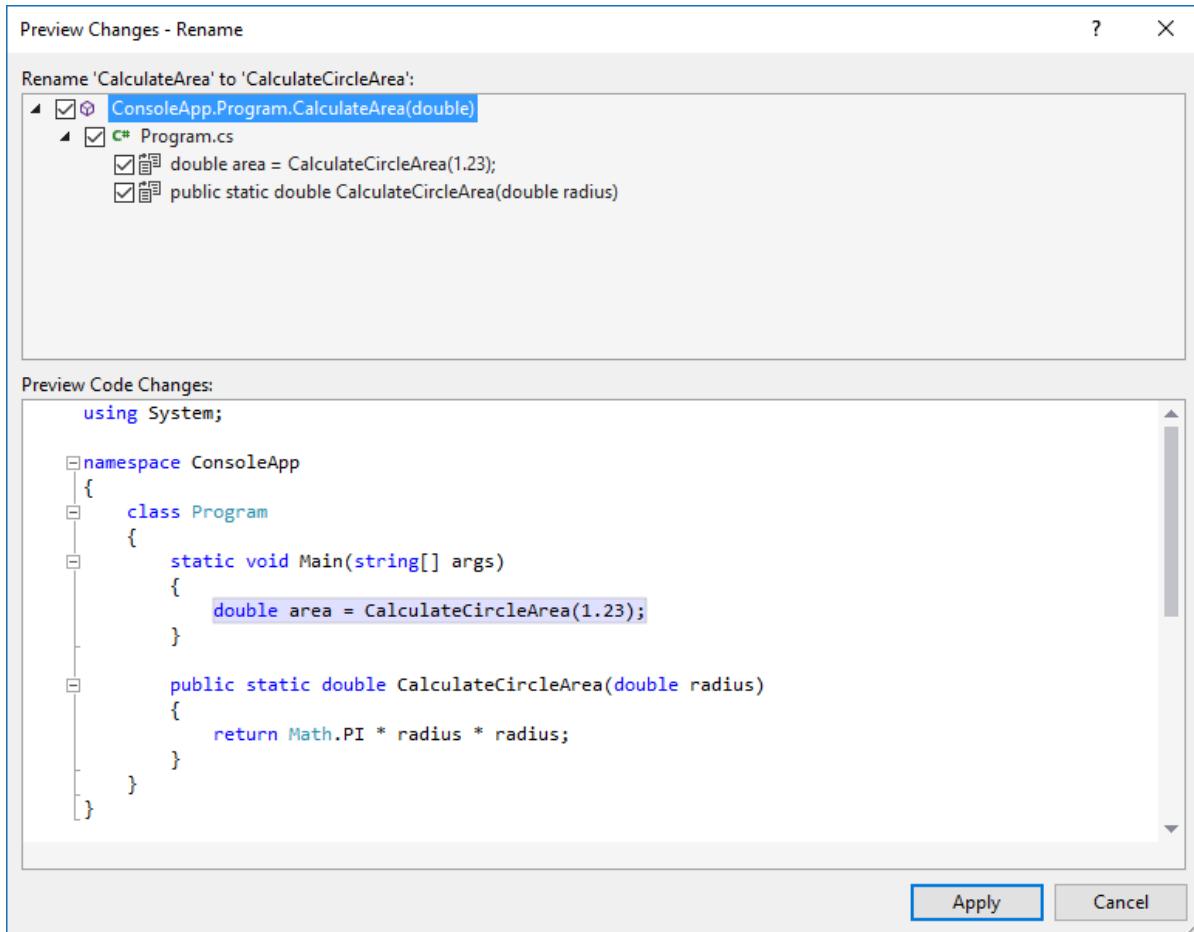
The following table describes several shortcut menu options that are available when you right-click a node in the tree view pane.

| CONTEXT MENU ITEM | DESCRIPTION |
|----------------------------|--|
| Add As New Root | Adds the selected node to the tree view pane as a new root node. This enables you to focus your attention on a specific subtree. |
| Remove Root | Removes the selected root node from the tree view pane. This option is available only from a root node. You can also use the Remove Root toolbar button to remove the selected root node. |
| Go To Definition | Runs the Go To Definition command on the selected node. This navigates to the original definition for a member call or variable definition. To run the Go To Definition command, you can also double-click the selected node or press F12 on the selected node. |
| Find All References | Runs the Find All References command on the selected node. This finds all the lines of code in your project that reference a class or member. You can also use SHIFT+F12 to run the Find All References command on the selected node. |
| Copy | Copies the contents of the selected node (but not its subnodes). |
| Refresh | Collapses the selected node so that re-expanding it displays current information. |

Preview Changes window

1/13/2018 • 1 min to read • [Edit Online](#)

When using various *Quick Actions* or *Refactoring* tools in Visual Studio, it is often possible to preview changes that are going to be made to your project prior to accepting them. The **Preview Changes** window is where this is done. For example, here is the **Preview Changes** window showing what will be changed during a Rename refactor in a C# project:



The top half of the window shows the specific lines that will be changed, each with a checkbox. You can check or uncheck each checkbox if you want to selectively apply the refactoring to only specific lines.

The bottom half of the window shows the formatted code from the project that will be changed, with the affected areas highlighted. Selecting the specific line in the top half of the window will highlight the corresponding line in the bottom half. This allows you to quickly skip to the appropriate line and see the surrounding code.

After reviewing the changes, click the **Apply** button to commit those changes, or click the **Cancel** button to leave things as they were.

See also

[Refactoring in Visual Studio](#)

[Quick Actions](#)

Choose Toolbox items, WPF components

1/22/2018 • 2 min to read • [Edit Online](#)

This tab of the **Choose Toolbox Items** dialog box displays a list of Windows Presentation Foundation (WPF) controls available on your local computer. To display this list, select **Choose Toolbox Items** from the **Tools** menu to display the **Choose Toolbox Items** dialog box, and then select its **WPF Components** tab. To sort the components listed, select any column heading.

- When the check box next to a component is selected, an icon for that component will be displayed in the **Toolbox**.

TIP

To add a WPF control to a project document that's open for editing, drag its **Toolbox** icon onto the Design view surface. Default markup and code for the component are inserted into your project, ready for you to modify. For more information, see [Toolbox](#).

- When the check box next to a component is cleared, the corresponding icon will be removed from **Toolbox**.

NOTE

The .NET Framework components installed on your computer remain available whether or not icons for them are displayed in the **Toolbox**.

The columns on the **WPF Components** tab contain the following information:

Name

Lists the names of WPF controls for which entries exist in your computer's registry.

Namespace

Displays the hierarchy of the [.NET Framework Class API](#) namespace that defines the structure of the component. Sort on this column to list the available components within each .NET Framework namespace installed on your computer.

Assembly Name

Displays the name of the .NET Framework assembly that includes the namespace for each component. Sort on this column to list the namespaces contained in each .NET Framework assembly installed on your computer.

Directory

Displays the location of the .NET Framework assembly. The default location for all assemblies is the Global Assembly Cache. For further information on the Global Assembly Cache, see [Working with Assemblies and the Global Assembly Cache](#).

UIElement List

Filter

Filters the list of WPF controls based on the string you provide in the text box. All matches from any of the four columns are shown.

Clear

Clears the filter string.

Browse

Opens the **Open** dialog box, which lets you navigate to assemblies which contain WPF controls. Use this to load assemblies which are not located in the Global Assembly Cache.

Language

Shows the localized language of the assembly which contains the selected WPF control.

Limitations

Adding a custom control or [UserControl](#) to the Toolbox has the following limitations:

- Works only for custom controls defined outside the current project.
- Does not update correctly when you change the solution configuration from Debug to Release, or from Release to Debug. This is because the reference is not a project reference, but is for the assembly on disk instead. If the control is part of the current solution, when you change from Debug to Release, your project continues to reference the Debug version of the control.

In addition, if design-time metadata is applied to the custom control and this metadata specifies that the [Microsoft.Windows.Design.ToolboxBrowsableAttribute](#) is set to `false`, the control does not appear in the Toolbox.

You can reference your controls directly in XAML view by mapping the namespace and assembly for your control.

See also

[Toolbox](#)

[Getting Started with WPF](#)

Code Snippet Picker

2/6/2018 • 1 min to read • [Edit Online](#)

The Visual Studio Code Editor provides a **Code Snippet Picker** that allows you, in a few mouse clicks, to insert ready-made blocks of code into the active document.

The procedure to display the **Code Snippet Picker** varies according to the language you are using.

- Visual Basic - Right-click at the desired location in the Code Editor to display the Shortcut menu, and select **Insert Snippet**.
- C# - Right-click at the desired location in the Code Editor to display the Shortcut menu, and click **Insert Snippet or Surround With**.
- C++ - The **Code Snippet Picker** is not available.
- F# - The **Code Snippet Picker** is not available.
- JavaScript - Right-click at the desired location in the Code Editor to display the Shortcut menu, and click **Insert Snippet or Surround With**.
- XML - Right-click at the desired location in the Code Editor to display the Shortcut menu, and click **Insert Snippet or Surround With**.
- HTML - Right-click at the desired location in the Code Editor to display the Shortcut menu, and click **Insert Snippet or Surround With**.
- SQL - Right-click at the desired location in the Code Editor to display the Shortcut menu, and click **Insert Snippet**.

In most Visual Studio development languages, you can use the **Code Snippets Manager** to add folders to the folder list that the **Code Snippet Picker** scans for XML snippet files. You can also create your own snippets to add to the list. For more information, see [Walkthrough: Creating a code snippet](#).

UIElement List

Item Name

An editable text field that displays the name of the item selected in the **Item List**. To perform an incremental search for the item you want, begin typing its name in this field. Continue adding letters until the desired item is selected in the **Item List**.

Item List

A list of code snippets available for insertion, or a list of folders containing code snippets. To insert a snippet or expand a folder, select the item you want and press Enter.

See also

[Best Practices for Using Code Snippets](#)

[Visual Basic IntelliSense Code Snippets](#)

[Setting Bookmarks in Code](#)

[How to: Use Surround-with Code Snippets](#)

Command Window

12/22/2017 • 3 min to read • [Edit Online](#)

The **Command** window is used to execute commands or aliases directly in the Visual Studio integrated development environment (IDE). You can execute both menu commands and commands that do not appear on any menu. To display the **Command** window, choose **Other Windows** from the **View** menu, and select **Command Window**.

Displaying the Values of Variables

To check the value of a variable `varA`, use the [Print Command](#):

```
>Debug.Print varA
```

The question mark (?) is an alias for `Debug.Print`, so this command can also be written:

```
>? varA
```

Both versions of this command will return the value of the variable `varA`.

Entering Commands

The greater than symbol (>) appears at the left edge of the Command window as a prompt for new lines. Use the UP ARROW and DOWN ARROW keys to scroll through previously issued commands.

| TASK | SOLUTION | EXAMPLE |
|---|--|---------|
| Evaluate an expression. | Preface the expression with a question mark (?). | ? myvar |
| Switch to an Immediate window. | Enter <code>immed</code> into the window without the greater than sign (>) | immed |
| Switch back to the Command window from an Immediate window. | Enter <code>cmd</code> into the window. | >cmd |

The following shortcuts help you navigate while in Command mode.

| ACTION | CURSOR LOCATION | KEYBINDING |
|--|-------------------------|-------------------------------|
| Cycle through the list of previously entered commands. | Input line | UP ARROW & DOWN ARROW |
| Scroll up the window. | Command window contents | CTRL+UP ARROW |
| Scroll down the window. | Command window contents | DOWN ARROW or CTRL+DOWN ARROW |

TIP

You can copy all or part of a previous command to the input line by scrolling to it, highlighting all or part of it, and then pressing ENTER.

Mark Mode

When you click on any previous line in the **Command** window, you shift automatically into Mark mode. This allows you to select, edit, and copy the text of previous commands as you would in any text editor, and paste them into the current line.

The Equals (=) Sign

The window used to enter the `EvaluateStatement` command determines whether an equals sign (=) is interpreted as a comparison operator or as an assignment operator.

In the **Command** window, an equals sign (=) is interpreted as a comparison operator. You cannot use assignment operators in the **Command** window. So, for example, if the values of variables `varA` and `varB` are different, then the command `>Debug.EvaluateStatement(varA=varB)` will return a value of `False`.

In the **Immediate** window, by contrast, an equals sign (=) is interpreted as an assignment operator. So, for example, the command `>Debug.EvaluateStatement(varA=varB)` will assign to variable `varA` the value of variable `varB`.

Parameters, Switches, and Values

Some Visual Studio commands have required and optional arguments, switches and values. Certain rules apply when dealing with such commands. The following is an example of a rich command to clarify the terminology.

```
Edit.ReplaceInFiles /case /pattern:regex var[1-3]+ oldpar
```

In this example,

- `Edit.ReplaceInFiles` is the command
- `/case` and `/pattern:regex` are switches (prefaced with the slash [/] character)
- `regex` is the value of the `/pattern` switch; the `/case` switch has no value
- `var[1-3]+` and `oldpar` are parameters

NOTE

Any command, parameter, switch, or value that contains spaces must have double quotation marks on either side.

The position of switches and parameters can be interchanged freely on the command line with the exception of the [Shell](#) command, which requires its switches and parameters in a specific order.

Nearly every switch supported by a command has two forms: a short (one character) form and a long form. Multiple short-form switches can be combined into a group. For example, `/p /g /m` can be expressed alternately as `/pgm`.

If short-form switches are combined into a group and given a value, that value applies to every switch. For example, `/pgm:123` equates to `/p:123 /g:123 /m:123`. An error occurs if any of the switches in the group does not accept a value.

Escape Characters

A caret (^) character in a command line means that the character immediately following it is interpreted literally, rather than as a control character. This can be used to embed straight quotation marks ("), spaces, leading slashes, carets, or any other literal characters in a parameter or switch value, with the exception of switch names. For example,

```
>Edit.Find ^^t /regex
```

A caret functions the same whether it is inside or outside quotation marks. If a caret is the last character on the line, it is ignored. The example shown here demonstrates how to search for the pattern "`^t`".

Use Quotes for Path Names with Spaces

If, for example, you want to open a file that has a path containing spaces, you must put double quotes around the path or path segment that contains spaces: **C:\Program Files** or **"C:\Program Files"**.

See Also

[Visual Studio Command Aliases](#)

[Visual Studio Commands](#)

Convert Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

The **Convert** dialog box allows you to select a conversion tool to create a new project from an old one, such as creating a Visual Basic 2005 project from a Visual Basic 6 project. You can also use the [Visual Studio Project Converter](#) to upgrade projects created in previous versions of Visual Studio.

Opening the Convert Dialog Box

To access the Convert dialog box

1. On the menu bar, choose **File**, **Open**, **Convert**.

You will see a list of the code-conversion tools that are currently available.

2. Select the tool that is appropriate for your application.

See Also

[Porting, Migrating, and Upgrading Visual Studio Projects](#)

Error List Window

12/22/2017 • 3 min to read • [Edit Online](#)

NOTE

The Error List displays information about a specific error message. You can copy the error number or error string text from the Output window. To display the Output window, press Ctrl+Alt+O. See [Output Window](#).

You can develop apps faster by using the **Error List** window. For example, you can perform the following tasks:

- Display the errors, warnings, and messages produced while you write code.
- Find syntax errors noted by IntelliSense.
- Find deployment errors, certain Static Analysis errors, and errors detected while applying Enterprise Template policies.
- Double-click any error message entry to open the file where the problem occurs, and move to the error location.
- Filter which entries are displayed, and which columns of information appear for each entry.
- Search for specific terms and scope the search to just the current project or document.

To display the **Error List**, click **View / Error List**, or **CTRL+\+E**.

You can choose the **Errors**, **Warnings**, and **Messages** tabs to see different levels of information.

To sort the list, click any column header. To sort again by an additional column, hold down the SHIFT key and click another column header. To select which columns are displayed and which are hidden, choose **Show Columns** from the shortcut menu. To change the order in which columns are displayed, drag any column header to the left or right.

NOTE

The dialog boxes and menu commands you see might differ from those described here, depending on your active settings or edition. To change your settings, click **Tools / Import and Export Settings**. For more information, see [Personalize the Visual Studio IDE](#).

Error List Filters

There are two types of filter in two dropdown boxes, one on the right side of the toolbar and one to the left of the toolbar. The dropdown list on the left side of the toolbar specifies the set of code files to use (**Entire Solution**, **Open Documents**, **Current Project**, **Current Document**).

You can restrict the scope of the search to analyze and act on groups of errors. For example, you might want to focus on core errors that are preventing a project from compiling. The scoping options include:

1. **Open Documents:** Show errors, warnings, and messages for the open documents.
2. **Current Project:** Show errors, warnings, and messages from the project of the currently selected document in the **Editor** or the selected project in **Solution Explorer**.

NOTE

The filtered list of errors, warnings, and messages will change if the project of the currently selected document is different from the project selected in **Solution Explorer**.

3. **Current Document:** Show errors, warnings, and messages for the currently selected document in the **Editor** or **Solution Explorer**.

If a filter is currently applied to the search result, the name of the filter appears in the **Error List** title bar. The **Errors**, **Warnings**, and **Messages** buttons then display the number of filtered items being shown along with the total number of items; for example, the buttons show x of y Errors. If no filter is applied, the title bar says only "Error List".

The list on the right side of the toolbar specifies whether to show errors from the build (errors resulting from a build operation) or from IntelliSense (errors detected before running a build), or from both.

Search

Use the **Search Error List** text box on the right side of the **Error List** toolbar to find specific errors in the error list. You can search on any visible column in the error list, and search results are always sorted based on the column that has sort priority instead of on the query or the filter applied. If you choose the **Esc** key while the focus is in the **Error List**, you can clear the search term and filtered search results. You can also click the **X** on the right side of the text box to clear it.

Save

You can copy the error list and save it to a file. Select the errors you want to copy and right-click the selection, then on the context menu select **Copy**. You can then paste the errors into a file. If you paste the errors to an Excel spreadsheet, the fields appear as different columns.

UI Element List

Severity

Displays the different types of **Error List** entry (**Error**, **Message**, **Warning**, **Warning (active)**, **Warning (inactive)**).

Code

Displays the error code.

Description

Displays the text of the entry.

Project

Displays the name of the current project.

File

Displays the file name.

Line

Displays the line where the problem occurs.

File Properties, JavaScript

12/22/2017 • 2 min to read • [Edit Online](#)

You can use file properties to indicate what actions the project system should perform on the files. For example, you can set file properties to indicate whether a file should be added to the package as a resource file.

You can select any file in Solution Explorer and then examine its properties in the Properties window. JavaScript files have four properties: **Copy to Output Directory**, **Package Action**, **File Name**, and **File Path**.

File Properties

This section describes properties common to JavaScript files.

Copy to Output Directory Property

This property specifies the conditions under which the selected source file will be copied to the output directory. Select **Do not copy** if the file is never to be copied to the output directory. Select **Copy always** if the file is always to be copied to the output directory. Select **Copy if newer** if the file is to be copied only when it is newer than an existing file of the same name in the output directory.

Package Action

The **Package Action** property indicates what Visual Studio does with a file when a build is executed. **Package Action** can have one of several values:

- **None** - The file is not included in the package manifest. An example is a text file that contains documentation, such as a Readme file.
- **Content** - The file is included in the package manifest. For example, this setting is the default value for an .htm, .js, .css, image, audio, or video file.
- **Manifest** - The file is not included in the package manifest. Instead, the file is used for input when generating the package manifest. This is the default value for the package.appxmanifest file.
- **Resource** - The file is not included in the package manifest. Instead, the contents of the file are indexed in the Package Resource Index (PRI) that goes into the package manifest. It is typically used for resource files.

The default value for **Package Action** depends on the extension of the file that you add to the solution.

File Name Property

Displays the file name as a read-only value. To rename the file, you must right-click in Solution Explorer and select **Rename**.

Full Path Property

Displays the full path to the file as a read-only value. To change the path of the file, you can drag-and-drop the file in Solution Explorer.

Reference File Properties

This section describes properties common to files referenced from a UWP app built using JavaScript. When you select a reference such as a .winmd file, an SDK reference, a project-to-project reference, or an assembly reference in Solution Explorer, other properties may display in the Properties window, according to the file type.

Culture

Displays the language associated with the reference.

File Type

Displays the file type of the reference.

File Version

Displays the file version of the reference.

Identity

Displays the identity of the reference that is used in the project, which is stored in the project file.

Package

Displays the name of the package manifest associated with the reference.

Resolved Path

Displays the path to the reference that is used in the project.

SDK Path

Displays the path to the referenced SDK file.

Uri

Displays the URI that must be included in the project's HTML or JavaScript files to include the file as a source file.

Version

Displays the version of the reference.

See Also

[Managing Project and Solution Properties](#)

Go To Line

12/22/2017 • 1 min to read • [Edit Online](#)

This dialog box allows you to move to a specific line in the active document. To access this dialog box, open a document for editing, then select **Go To** on the **Edit** menu.

Line number (1 -)

Allows you to enter the number of the line in the active document to which you want to move. The number entered must fall between 1 and , the number of lines in the current document.

See Also

[Setting Bookmarks in Code](#)

[Finding and Replacing Text](#)

[Writing Code](#)

Immediate Window

12/22/2017 • 3 min to read • [Edit Online](#)

The **Immediate** window is used to debug and evaluate expressions, execute statements, print variable values, and so forth. It allows you to enter expressions to be evaluated or executed by the development language during debugging. To display the **Immediate** window, open a project for editing, then choose **Windows** from the **Debug** menu and select **Immediate**, or press CTRL+ALT+I.

You can use this window to issue individual Visual Studio commands. The available commands include `EvaluateStatement`, which can be used to assign values to variables. The **Immediate** window also supports IntelliSense.

Displaying the Values of Variables

This window can be particularly useful while debugging an application. For example, to check the value of a variable `varA`, you can use the [Print Command](#):

```
>Debug.Print varA
```

The question mark (?) is an alias for `Debug.Print`, so this command can also be written:

```
>? varA
```

Both versions of this command will return the value of the variable `varA`.

NOTE

To issue a Visual Studio command in the **Immediate** window, you must preface the command with a greater than sign (>). To enter multiple commands, switch to the **Command** window.

Design Time Expression Evaluation

You can use the **Immediate** window to execute a function or subroutine at design time.

To execute a function at design time

1. Copy the following code into a Visual Basic console application:

```
Module Module1

    Sub Main()
        MyFunction(5)
    End Sub

    Function MyFunction(ByVal input as Integer) As Integer
        Return input * 2
    End Function

End Module
```

2. On the **Debug** menu, click **Windows**, and then click **Immediate**.

3. Type `?MyFunction(2)` in the **Immediate** window and press Enter.

The **Immediate** window will run `MyFunction` and display `4`.

If the function or subroutine contains a breakpoint, Visual Studio will break execution at the appropriate point. You can then use the debugger windows to examine your program state. For more information see [Walkthrough: Debugging at Design Time](#).

You cannot use design time expression evaluation in project types that require starting up an execution environment, including Visual Studio Tools for Office projects, Web projects, Smart Device projects, and SQL projects.

Design Time Expression Evaluation in Multi-Project Solutions

When establishing the context for design time expression evaluation, Visual Studio references the currently selected project in Solution Explorer. If no project is selected in Solution Explorer, Visual Studio attempts to evaluate the function against the startup project. If the function cannot be evaluated in the current context, you will receive an error message. If you are attempting to evaluate a function in a project that is not the startup project for the solution and you receive an error, try selecting the project in Solution Explorer and attempt the evaluation again.

Entering Commands

You must enter the greater than sign (>) when issuing Visual Studio commands in the **Immediate** window. Use the UP ARROW and DOWN ARROW keys to scroll through previously issued commands.

| TASK | SOLUTION | EXAMPLE |
|---|--|------------------------|
| Evaluate an expression. | Preface the expression with a question mark (?). | <code>? a+b</code> |
| Temporarily enter Command mode while in Immediate mode (to execute a single command). | Enter the command, prefacing it with a greater than sign (>). | <code>>alias</code> |
| Switch to the Command window. | Enter <code>cmd</code> into the window, prefacing it with a greater than sign (>). | <code>>cmd</code> |
| Switch back to the Immediate window. | Enter <code>immed</code> into the window without the greater than sign (>). | <code>immed</code> |

Mark Mode

When you click on any previous line in the **Immediate** window, you shift automatically into Mark mode. This allows you to select, edit, and copy the text of previous commands as you would in any text editor, and paste them into the current line.

The Equals (=) Sign

The window used to enter the `EvaluateStatement` command determines whether an equals sign (=) is interpreted as a comparison operator or as an assignment operator.

In the **Immediate** window, an equals sign (=) is interpreted as an assignment operator. So, for example, the command

```
>Debug.EvaluateStatement(varA=varB)
```

will assign to variable `varA` the value of variable `varB`.

In the **Command** window, by contrast, an equals sign (=) is interpreted as a comparison operator. You cannot use assignment operations in the **Command** window. So, for example, if the values of variables `varA` and `varB` are different, then the command

```
>Debug.EvaluateStatement(varA=varB)
```

will return a value of `False`.

First-Chance Exception Notifications

In some settings configurations, first-chance exception notifications are displayed in the **Immediate** window.

To toggle first-chance exception notifications in the Immediate window

1. On the **View** menu, click **Other Windows**, and click **Output**.
2. Right-click on the text area of the **Output** window, and select or deselect **Exception Messages**.

See Also

[Navigating through Code with the Debugger](#)

[Command Window](#)

[Debugging in Visual Studio](#)

[Debugger Basics](#)

[Walkthrough: Debugging at Design Time](#)

[Visual Studio Command Aliases](#)

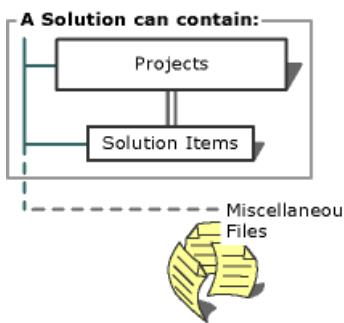
[Using Regular Expressions in Visual Studio](#)

Miscellaneous Files

12/22/2017 • 2 min to read • [Edit Online](#)

You might want to use the Visual Studio editors to work independently on files from a project or from a solution. While you have a solution open, you can open and modify files without adding them to a solution or to a project. Files you want to work with independently from the containers are called miscellaneous files. Miscellaneous files are external to solutions and projects, are not included in builds, and cannot be included with a solution under source control.

Opening files independently from a container is useful for a variety of reasons. You might have a file that you want to view while developing a project-based solution but that is not integral to the solution's development. Common examples include development notes or instructions, database schema, and code clips. In addition, you might want to create a stand-alone file.



Solution Explorer can display a Miscellaneous Files folder for the files if the options for the folder are enabled. The options can be set from the [Documents, Environment, Options Dialog Box](#). After you close a miscellaneous file, it is not associated with any particular solution or project unless an option is enabled for that as well.

The Miscellaneous Files folder represents the files as links. Although this folder is not part of a solution, when you open a solution, some or all of the miscellaneous files that were opened when the solution was last closed are re-opened, depending upon the settings for the folder.

NOTE

Some of the files that do not appear in the Miscellaneous Files folder are files that you cannot modify within the IDE, such as .zip files and .doc files. The IDE will not track files that can only be modified through an external editor.

Commands Available in the IDE

The menus, toolbars, and the commands they contain change based on the format of the file you open. When you open a text file, for example, the Text Editor toolbar appears and its commands are available. If you then open an XML Schema file, the XML Schema toolbar appears. While editing your XML Schema, the Text Editor toolbar's commands (or the toolbar itself) are unavailable. The XML Schema is the active window and as such, has current selection context. When you switch between a project file and a miscellaneous file, all project-related commands disappear and only those that are directly related to the miscellaneous file appear.

Folder Display Options

You can set display options for the Miscellaneous Folder so that the folder appears even though you have not opened any Miscellaneous Files. The solution file does not permanently manage a list of miscellaneous files. It uses an optional feature that allows it to remember a per-user most recently used (MRU) list of files.

See Also

[Solutions and Projects](#)

[Documents, Environment, Options Dialog Box](#)

Options Dialog Box (Visual Studio)

12/22/2017 • 2 min to read • [Edit Online](#)

The **Options** dialog box enables you to configure the integrated development environment (IDE) to your needs. For example, you can establish a default save location for your projects, alter the default appearance and behavior of windows, and create shortcuts for commonly used commands. There are also options specific to your development language and platform. You can access **Options** from the **Tools** menu.

NOTE

The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Layout of the Options dialog box

The **Options** dialog box is divided into two parts: a navigation pane on the left and a display area on the right. The tree control in the navigation pane includes folder nodes, such as Environment, Text Editor, Projects and Solutions, and Source Control. Expand any folder node to list the pages of options that it contains. When you select the node for a particular page, its options appear in the display area.

Options for an IDE feature do not appear in the navigation pane until the feature is loaded into memory. Therefore, the same options might not be displayed as you begin a new session that were displayed as you ended the last. When you create a project or run a command that uses a particular application, nodes for relevant options are added to the Options dialog box. These added options will then remain available as long as the IDE feature remains in memory.

NOTE

Some settings collections scope the number of pages that appear in the navigation pane of the Options dialog box. You can choose to view all possible pages by selecting **Show all settings**.

How options are applied

Clicking **OK** in the **Options** dialog box saves all settings on all pages. Clicking **Cancel** on any page cancels all change requests, including any just made on other **Options** pages. Some changes to option settings, such as those made on [Fonts and Colors](#), [Environment](#), [Options Dialog Box](#), will only take effect after you close and reopen Visual Studio.

Show all settings

Selecting or unselecting **Show all settings** applies all changes you have made in the **Options** dialog box, even though you have not yet clicked **OK**.

See Also

[Customizing the Editor](#)

Environment Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

You can use the pages in the **Environment** folder in the **Options** dialog box to set how certain elements of the integrated development environment (IDE) display and behave. You can access the Environment pages by, on the menu bar, choosing **Tools**, **Options**, and then selecting **Environment**.

NOTE

The dialog boxes and menu commands that appear on your computer might differ from those that are described in Help, depending on your active settings and the edition of Visual Studio that you are using. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

| FOR INFORMATION ABOUT HOW TO | SEE |
|---|---|
| Customize the appearance and behavior of windows, menus, and other elements of the IDE. | General, Environment, Options Dialog Box |
| Change the settings for automatic saving and restoration of files. | AutoRecover, Environment, Options Dialog Box |
| Establish document display and management settings that include the behavior of Miscellaneous files. | Documents, Environment, Options Dialog Box |
| Customize how updates occur and how extensions are managed. | Extensions and Updates, Environment, Options Dialog Box |
| Customize when message boxes appear and how the Find and Replace window acts during a Quick Find or Quick Replace, and also auto-populate Find What. | Find and Replace, Environment, Options Dialog Box |
| Set the font and color in certain editors, dialog boxes, tools windows, and other user interface (UI) elements. | Fonts and Colors, Environment, Options Dialog Box |
| Customize the location of vssettings files that store user setting information. | Import and Export Settings, Environment, Options Dialog Box |
| Select a default language for Visual Studio. | International Settings, Environment, Options Dialog Box |
| To set keyboard mapping options. | Keyboard, Environment, Options Dialog Box |
| To specify whether notifications are enabled. | Notifications, Environment, Options Dialog Box |
| To specify options for Quick Launch window. | Quick Launch, Environment, Options Dialog Box |
| To specify start page options. | Startup, Environment, Options Dialog Box |
| To specify whether to roam settings across machines. | Synchronized Settings, Environment, Options Dialog Box |
| Learn how to set options for tasks and task lists | Task List, Environment, Options Dialog Box |

| FOR INFORMATION ABOUT HOW TO | SEE |
|--|--|
| Change your default Home and Search pages, changing the system source editor, and configuring Internet Explorer options. | Web Browser , Environment , Options Dialog Box |

See Also

[Options Dialog Box](#)

General, Environment, Options Dialog Box

3/12/2018 • 2 min to read • [Edit Online](#)

Use this page to change color themes, status bar settings, and file extension associations, among other options, for the integrated development environment (IDE). You can access the **Options** dialog box by opening the **Tools** menu, choosing **Options**, opening the **Environment** folder and then choosing the **General** page. If this page does not appear in the list, select the **Show all settings** check box in the **Options** dialog box.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, open the **Tools** menu, and then choose **Import and Export Settings**. For more information, see [Personalize the Visual Studio IDE](#).

Visual Experience

Color theme

Choose the **Blue**, **Light** or **Dark** color theme for the IDE.

You can install additional predefined themes, and create custom themes, by downloading and installing the **Visual Studio Color Theme Editor** from the [Visual Studio Marketplace](#). After you install this tool, additional color themes appear in the Color theme list box.

Apply title casing in menu bar

Menus are in **Title Casing** by default. Un-check this option to set them to **ALL CAPS**.

Automatically adjust visual experience based on client performance

Specifies whether Visual Studio sets the adjustment to the visual experience automatically or you set the adjustment explicitly. This adjustment may change the display of colors from gradients to flat colors, or it may restrict the use of animations in menus or popup windows.

Enable rich client experience

Enables the full visual experience of Visual Studio, including gradients and animations. Clear this option when using Remote Desktop connections or older graphics adapters, because these features may have poor performance in those cases. This option is available only when you clear the **Automatically adjust visual experience based on client** option.

Use hardware graphics acceleration if available

Uses hardware graphics acceleration if it is available, rather than software acceleration.

Other

Items shown in Window menu Customizes the number of windows that appear in the Windows list of the **Window** menu. Type a number between 1 and 24. By default, the number is 10.

Items shown in recently used lists Customizes the number of most recently used projects and files that appear on the **File** menu. Type a number between 1 and 24. By default, the number is 10. This is an easy way to retrieve recently used projects and files.

Show status bar Displays the status bar. The status bar is located at the bottom of the IDE window and displays information about the progress of ongoing operations.

Close button affects active tool window only Specifies that when the **Close** button is clicked, only the tool window that has focus is closed and not all of the tool windows in the docked set. By default, this option is selected.

Auto Hide button affects active tool window only Specifies that when the **Auto Hide** button is clicked, only the tool window that has focus is hidden automatically and not all of the tool windows in the docked set. By default, this option is not selected.

See also

[Environment Options Dialog Box Customizing window layouts](#)

AutoRecover, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Use this page of the Options dialog box to specify whether or not files are automatically backed up. This page also allows you to specify whether or not modified files are restored when the integrated development environment (IDE) shuts down unexpectedly. You can access this dialog box by selecting the **Tools** menu and choosing **Options**, and then selecting the **Environment** folder and choosing the **AutoRecover** page. If this page does not appear in the list, select **Show all settings** in the **Options** dialog box.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see [Personalize the Visual Studio IDE](#).

Save AutoRecover information every <n> minutes

Use this option to customize how often a file is automatically saved in the editor. For previously saved files, a copy of the file is saved in `\...\My Documents\Visual Studio <version>\Backup Files\<projectname>`. If the file is new and has not been manually saved, the file is auto-saved using a randomly generated file name.

Keep AutoRecover information for <n> days

Use this option to specify how long Visual Studio keeps files created for autorecovery.

See Also

[Options Dialog Box](#)

Documents, Environment, Options Dialog Box

12/22/2017 • 3 min to read • [Edit Online](#)

Use this page of the **Options** dialog box to control the display of documents in the integrated development environment (IDE) and manage external changes to documents and files. You can access this dialog box by clicking **Options** on the **Tools** menu and then selecting **Documents** in the **Environment** node. If **Documents** does not appear in the list, select **Show all settings** in the **Options** dialog box.

NOTE

The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Reuse current document window, if saved

When selected, closes your current document if it has been saved, and opens a new document in the same window. If your current document has not been saved, it remains open and the new document is opened in a separate window. When this option is cleared, new documents always open in separate windows.

If you perform multi-document cut and paste operations infrequently and want to minimize the number of open documents and windows in your working space, try this option.

Detect when file is changed outside the environment

When this option is selected, a message immediately notifies you of changes to an open file that have been made by an editor outside the IDE. The message lets you reload the file from storage.

Auto-load changes, if saved

When you have **Detect when file is changed outside the environment** selected and an open file in the IDE changes outside the IDE, a warning message is generated by default. If this option is enabled, no warning appears and the document is reloaded in the IDE to pick up the external changes.

Allow editing of read-only files; warn when attempt to save

When this option is enabled you can open and edit a read-only file. When you are finished, you must use the **Save As** command to save the file by a new name if you want to save a record of your changes.

Open file using directory of currently active document

When selected, this option specifies that the **Open File** dialog box displays the directory of the active document. When this option is cleared, the **Open File** dialog box displays the directory last used to open a file.

Check for consistent line endings on load

Select this option to have the editor scan the line endings in a file and display a message box if inconsistencies are detected in how line endings are formatted.

Display warning when global undo will modify edited files

Select this option to display a message box when the **Global Undo** command will roll back refactoring changes made in files that also were changed after the refactoring operation. Returning a file to its pre-refactoring state might discard subsequent changes made in the file.

Show Miscellaneous files in Solution Explorer

Select this option to display the **Miscellaneous Files** node in **Solution Explorer**. Miscellaneous files are files that are not associated with a project or solution but can appear in **Solution Explorer** for your convenience.

NOTE

Select this option to enable the **View in Browser** command on the **File** menu for Web documents not included in the active Web application.

< n > items saved in the Miscellaneous files project

Specifies the number of files to persist in the **MiscellaneousFiles** folder of **Solution Explorer**. These files are listed even if they are no longer open in an editor. You can specify any whole number from 0 to 256. The default number is 0.

For example, if you set this option to 5 and have 10 miscellaneous files open, when you close all 10 files, the first 5 will still be shown in the **Miscellaneous Files** folder.

Save documents as Unicode when data cannot be saved in codepage

Select this option to cause files containing information incompatible with the selected codepage to be saved as Unicode by default.

See Also

[Environment Options Dialog Box](#)

[Miscellaneous Files](#)

[Finding and Replacing Text](#)

Extensions and Updates, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Use this page to set options for how Visual Studio performs updates and how extensions are updated and discovered.

Automatically check for updates

When checked, Visual Studio will check periodically for updates to itself, installed SDKs and tools, and extensions, and display notifications in the menu bar when updates are available. For more information, see [Notifications](#).

Automatically update extensions

When checked, updates to extensions are made without prompting. For more information, see [Finding and Using Visual Studio Extensions](#).

Load user extensions when running as administrator

For more information, see [Finding and Using Visual Studio Extensions](#).

Additional Extension Galleries

An Enterprise feature that enables support for galleries of proprietary extensions. For more information, see [Private Galleries](#).

See Also

[Environment Options Dialog Box](#)

Find and Replace, Environment, Options Dialog Box

12/22/2017 • 2 min to read • [Edit Online](#)

Use this page of the **Options** dialog box to control message boxes and other aspects of a find and replace operation. You can access this dialog box from the **Tools** menu by clicking **Options**, expanding **Environment**, and then clicking **Find and Replace**. If this page does not appear in the list, select **Show all settings** in the **Options** dialog box.

NOTE

The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

UIElement List

Display informational messages

Select this option to display all Find and Replace informational messages that have the **Always show this message** option. For example, if you chose not to display the message "Find reached the starting point of the search.", selecting this option would cause this informational message to appear again when you use Find and Replace.

If you do not want to see any informational messages for Find and Replace, clear this option.

When you have cleared the **Always show this message** option on some, but not all, **Find and Replace** informational messages, the **Display informational messages** check box appears to be filled but not selected. To restore all optional **Find and Replace** messages, clear this option and then select it again.

NOTE

This option does not affect any **Find and Replace** informational messages that do not display the **Always show this message** option.

Display warning messages

Select this option to display all cautionary Find and Replace messages that have the **Always show this message** option. For example, if you chose not to display the **Replace All** warning message that appears when you attempt to make replacements in files not currently opened for editing, selecting this option would cause this warning message to appear again when you attempt to Replace All.

If you do not want to see any cautionary messages for Find and Replace, clear this option.

When you have cleared the **Always show this message** option on some, but not all, **Find and Replace** warning messages, the **Display warning messages** check box appears to be filled but not selected. To restore all optional **Find and Replace** messages, clear this option and then select it again.

NOTE

This option does not affect any **Find and Replace** warning messages that do not display the **Always show this message** option.

Automatically populate Find What with text from the editor

Select this option to paste the text on either side of the current editor's insertion point into the **Find what** field when you select any view of the **Find and Replace** window from the **Edit** menu. Clear this option to use the last search pattern from the previous search as the **Find what** string.

See Also

[Finding and Replacing Text](#)

Fonts and Colors, Environment, Options Dialog Box

12/22/2017 • 16 min to read • [Edit Online](#)

The **Fonts and Colors** page of the **Options** dialog box lets you establish a custom font and color scheme for various user interface elements in the integrated development environment (IDE). You can access this dialog box by clicking **Tools / Options**, and then selecting **Environment / Fonts and Colors**. If this page does not appear in the list, select **Show all settings** in the **Options** dialog box.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Color scheme changes do not take effect during the session in which you make them. You can evaluate color changes by opening another instance of Visual Studio and producing the conditions under which you expect your changes to apply.

Show settings for

Lists all of the user interface elements for which you can change font and color schemes. After selecting an item from this list you can customize color settings for the item selected in **Display items**.

- **Text Editor**

Changes to font style, size, and color display settings for Text Editor affect the appearance of text in your default text editor. Documents opened in a text editor outside the IDE will not be affected by these settings.

- **Printer**

Changes to font style, size, and color display settings for Printer affect the appearance of text in printed documents.

NOTE

As needed, you can select a different default font for printing than that used for display in the text editor. This can be useful when printing code that contains both single-byte and double-byte characters.

- **Statement Completion**

Changes the font style and size for the text that appears in statement completion pop-up in the editor.

- **Editor Tooltip**

Changes the font style and size for the text that appears in ToolTips displayed in the editor.

- **Environment Font**

Changes the font style and size for all IDE user interface elements that do not already have a separate option in **Show settings for**. For example, this option applies to the **Start Page** but would not affect the **Output** window.

- **[All Text Tool Windows]**

Changes to font style, size, and color display settings for this item affect the appearance of text in tool

windows that have output panes in the IDE. For example, Output window, Command window, Immediate window, etc.

NOTE

Changes to the text of **[All Text Tool Windows]** items do not take effect during the session in which you make them. You can evaluate such changes by opening another instance of Visual Studio.

Use Defaults

Resets the font and color values of the list item selected in **Show settings for**. The **Use** button appears when other display schemes are available for selection. For example, you can choose from two schemes for the Printer.

Font (**bold type** indicates fixed-width fonts)

Lists all the fonts installed on your system. When the drop-down menu first appears, the current font for the element selected in the **Show settings for** field is highlighted. Fixed fonts — which are easier to align in the editor — appear in bold.

Size

Lists available point sizes for the highlighted font. Changing the size of the font affects all **Display items** for the **Show settings for** selection.

Display items

Lists the items for which you can modify the foreground and background color.

NOTE

Plain Text is the default display item. As such, properties assigned to **PlainText** will be overridden by properties assigned to other display items. For example, if you assign the color blue to **PlainText** and the color green to **Identifier**, all identifiers will appear in green. In this example, **Identifier** properties override **PlainText** properties.

Some of display items include:

| DISPLAY ITEM | DESCRIPTION |
|-----------------------------------|---|
| Plain Text | Text in the editor. |
| Selected Text | Text that is included in the current selection when the editor has focus. |
| Inactive Selected Text | Text that is included in the current selection when the editor has lost focus. |
| Indicator Margin | The margin at the left of the Code Editor where breakpoints and bookmark icons are displayed. |
| Line Numbers | Optional numbers that appear next to each line of code |
| Visible White Space | Spaces, tabs and word wrap indicators |
| Bookmark | Lines that have bookmarks. Bookmark is only visible if the indicator margin is disabled. |
| Brace Matching (Highlight) | Highlighting that is typically bold formatting for matching braces. |

| DISPLAY ITEM | DESCRIPTION |
|---|---|
| Brace Matching (Rectangle) | Highlighting that is typically a grey rectangle in the background. |
| Breakpoint (Disabled) | Not used. |
| Breakpoint (Enabled) | Specifies the highlight color for statements or lines containing simple breakpoints. This option is applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Breakpoint (Error) | Specifies the highlight color for statements or lines containing breakpoints that are in an error state. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Breakpoint (Warning) | Specifies the highlight color for statements or lines containing breakpoints that are in a warning state. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Breakpoint - Advanced (Disabled) | Specifies the highlight color for statements or lines containing disabled conditional or hit-counted breakpoints. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Breakpoint - Advanced (Enabled) | Specifies the highlight color for statements or lines containing conditional or hit-counted breakpoints. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Breakpoint - Advanced (Error) | Specifies the highlight color for statements or lines containing conditional or hit-counted breakpoints that are in an error state. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Breakpoint - Advanced (Warning) | Specifies the highlight color for statements or lines containing conditional or hit-counted breakpoints that are in a warning state. Applicable only if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |

| DISPLAY ITEM | DESCRIPTION |
|--|--|
| Breakpoint - Mapped (Disabled) | Specifies the highlight color for statements or lines containing disabled mapped breakpoints. Applicable for ASP or ASP.NET debugging if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Breakpoint - Mapped (Enabled) | Specifies the highlight color for statements or lines containing mapped breakpoints. Applicable for ASP or ASP.NET debugging if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Breakpoint - Mapped (Error) | Specifies the highlight color for statements or lines containing mapped breakpoints in an error state. Applicable for ASP or ASP.NET debugging if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Breakpoint - Mapped (Warning) | Specifies the highlight color for statements or lines containing mapped breakpoints in a warning state. Applicable for ASP or ASP.NET debugging if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| C/C++ User Keywords | A constant within a particular code file defined by means of the <code>#define</code> directive. |
| Call Return | Specifies the highlight color for source statements or lines that indicate call return points when context is switched to a non-top stack frame when debugging. |
| Code Snippet Dependent Field | A field that will be updated when the current editable field is modified. |
| Code Snippet Field | Editable field when a code snippet is active. |
| Collapsible Text | A block of text or code that can be toggled in and out of view within the Code Editor. |
| Comment | Code comments. |
| Compiler Error | Blue squiggles in the editor indicating a compiler error. |
| Coverage Not Touched Area | Code that has not been covered by a unit test. |
| Coverage Partially Touched Area | Code that has been partially covered by a unit test. |
| Coverage Touched Area | Code that has been completely covered by a unit test. |

| DISPLAY ITEM | DESCRIPTION |
|--|---|
| CSS Comment | A comment in Cascading Style Sheets. For example: /* comment */ |
| CSS Keyword | Keywords in the Cascading Style Sheet. |
| CSS Property Name | The name of a property, such as Background. |
| CSS Property Value | The value assigned to a property such as blue. |
| CSS Selector | A string that identifies what elements the corresponding rule applies to. A selector can either be a simple selector, such as 'H1', or a contextual selector, such as 'H1 B', which consists of several simple selectors. |
| CSS String Value | A string in Cascading Style Sheets. |
| Current list location | Current line navigated to in a list tool window, such as the Output window or Find Results windows. |
| Current Statement | Specifies the highlight color for the source statement or line that indicates the current step position when debugging. |
| Debugger Data Changed | The color of text used to display changed data inside the Registers and Memory windows. |
| Definition Window Background | The background color of the Code Definition window. |
| Definition Window Current Match | The current definition in the Code Definition window. |
| Disassembly File Name | The color of text used to display file name breaks inside the Disassembly window. |
| Disassembly Source | The color of text used to display source lines inside the Disassembly window. |
| Disassembly Symbol | The color of text used to display symbol names inside the Disassembly window. |
| Disassembly Text | The color of text used to display op-code and data inside the Disassembly window. |
| Excluded Code | Code that is not to be compiled, per a conditional preprocessor directive such as <code>#if</code> . |
| Identifier | Identifiers in code such as the class names, methods names, and variable names. |
| Keyword | Keywords for the given language that are reserved. For example: class and namespace. |
| Memory Address | The color of text used to display the address column inside the Memory window. |

| DISPLAY ITEM | DESCRIPTION |
|------------------------------------|--|
| Memory Changed | The color of text used to display changed data inside the Memory window. |
| Memory Data | The color of text used to display data inside the Memory window. |
| Memory Unreadable | The color of text used to display unreadable memory areas within the Memory window. |
| Number | A number in code that represents an actual numeric value. |
| Operator | Operators such as +, -, and !=. |
| Other Error | Other error types not covered by other error squiggles. Currently, this includes rude edits in Edit and Continue. |
| Preprocessor Keyword | Keywords used by the preprocessor such as #include. |
| Read-Only Region | Code that cannot be edited. For example code displayed in the Code Definition View window or code that cannot be modified during Edit and Continue. |
| Refactoring Background | Background color of the Preview Changes dialog box. |
| Refactoring Current Field | Background color of the current element to be refactored in the Preview Changes dialog box. |
| Refactoring Dependent Field | Color of references of the element to be refactored in the Preview Changes dialog box. |
| Register Data | The color of text used to display data inside the Registers window. |
| Register NAT | The color of text used to display unrecognized data and objects inside the Registers window. |
| Smart Tag | Used to denote the outline when smart tags are invoked. |
| SQL DML Marker | Applies to the Transact-SQL editor. DML statements in this editor are marked with a bounding blue box by default. |
| Stale Code | Superseded code awaiting an update. In some cases, Edit and Continue cannot apply code changes immediately, but will apply them later as you continue debugging. This occurs if you edit a function that must call the function currently executing, or if you add more than 64 bytes of new variables to a function waiting on the call stack. When this happens, the debugger displays a "Stale Code Warning" dialog box, and the superseded code continues to execute until the function in question finishes and is called again. Edit and Continue applies the code changes at that time. |
| String | String literals. |

| DISPLAY ITEM | DESCRIPTION |
|---|---|
| String (C# @ Verbatim) | <p>String literals in C# that are interpreted verbatim. For example:</p> <pre data-bbox="822 276 874 300">@"x"</pre> |
| Syntax Error | Parse errors. |
| Task List Shortcut | If a Task List shortcut is added to a line, and the indicator margin is disabled, the line will be highlighted. |
| Tracepoint (Disabled) | Not used. |
| Tracepoint (Enabled) | Specifies the highlight color for statements or lines containing simple tracepoints. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Tracepoint (error) | Specifies the highlight color for statements or lines containing tracepoints that are in an error state. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Tracepoint (Warning) | Specifies the highlight color for statements or lines containing tracepoints that are in a warning state. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Tracepoint - Advanced (Disabled) | Specifies the highlight color for statements or lines containing disabled conditional or hit-counted tracepoints. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Tracepoint - Advanced (Enabled) | Specifies the highlight color for statements or lines containing conditional or hit-counted tracepoints. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Tracepoint - Advanced (Error) | Specifies the highlight color for statements or lines containing conditional or hit-counted tracepoints that are in an error state. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |

| DISPLAY ITEM | DESCRIPTION |
|--|--|
| Tracepoint - Advanced (Warning) | Specifies the highlight color for statements or lines containing conditional or hit-counted tracepoints that are in a warning state. This option is applicable only if statement-level tracepoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Tracepoint - Mapped (Disabled) | Specifies the highlight color for statements or lines containing disabled mapped tracepoints. Applicable for ASP or ASP.NET debugging if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Tracepoint - Mapped (Enabled) | Specifies the highlight color for statements or lines containing mapped tracepoints. Applicable for ASP or ASP.NET debugging if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Tracepoint - Mapped (Error) | Specifies the highlight color for statements or lines containing mapped tracepoints in an error state. Applicable for ASP or ASP.NET debugging if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Tracepoint - Mapped (Warning) | Specifies the highlight color for statements or lines containing mapped tracepoints in a warning state. Applicable for ASP or ASP.NET debugging if statement-level breakpoints are active or the Highlight entire source line for breakpoints or current statement option is selected on General, Debugging, Options Dialog Box . |
| Track Changes after save | Lines of code that have been modified since the file was opened but are saved to disk. |
| Track Changes before save | Lines of code that have been modified since the file was opened but are not saved to disk. |
| User Types | Types defined by users. |
| User Types (Delegates) | Type color for delegates. |
| User Types (Enums) | Type color used for enums. |
| User Types (Interfaces) | Type color for interfaces. |
| User Types (Value types) | Type color for value types such as structs in C#. |
| Visual Basic Read Only Marker | A marker specific to Visual Basic used for designating EnC, such as exception regions, a method definition, and non-leaf call frames. |

| DISPLAY ITEM | DESCRIPTION |
|-----------------------------------|---|
| Warning | Compiler warnings. |
| Warning Lines Path | Used for Static Analysis warning lines. |
| XML Attribute | Attribute names. |
| XML Attribute Quotes | The quote characters for XML attributes. |
| XML Attribute Value | Contents of XML attributes. |
| XML Cdata Section | Contents of <![CDATA[...]]>. |
| XML Comment | The contents of <!-- -->. |
| XML Delimiter | XML Syntax delimiters, including <, <?, <!, <!--, -->, ?, >, <![,]]>, and [,]. |
| XML Doc Attribute | The value of an xml documentation attribute, such as <param name="l"> where the "l" is colorized. |
| XML Doc Comment | The comments enclosed in the xml documentation comments. |
| XML Doc Tag | The tags in XML doc comments, such as /// <summary>. |
| XML Keyword | DTD keywords such as CDATA, IDREF, and NDATA. |
| XML Name | Element names and Processing Instructions target name. |
| XML Processing Instruction | Contents of Processing Instructions, not including target name. |
| XML Text | Plain text element content. |
| XSLT Keyword | XSLT element names. |

Item foreground

Lists the available colors you can choose for the foreground of the item selected in **Display items**. Because some items are related, and should therefore maintain a consistent display scheme, changing the foreground color of text also changes the defaults for elements such as Compiler Error, Keyword, or Operator.

Automatic Items can inherit the foreground color from other display items such as **Plain Text**. Using this option, when you change the color of an inherited display item, the color of the related display items also change automatically. For example, if you selected the **Automatic** value for **Compiler Error** and later changed the color of **Plain Text** to Red, **Compiler Error** would also automatically inherit the color Red.

Default The color that appears for the item the first time you start Visual Studio. Clicking the **Use Defaults** button resets to this color.

Custom

Displays the Color dialog box to allow you to set a custom color for the item selected in the Display items list.

NOTE

Your ability to define custom colors may be limited by the color settings for your computer display. For example, if your computer is set to display 256 colors and you select a custom color from the **Color** dialog box, the IDE defaults to the closest available **Basic color** and displays the color black in the **Color** preview box.

Item background

Provides a color palette from which you can choose a background color for the item selected in **Display items**.

Because some items are related, and should therefore maintain a consistent display scheme, changing the background color of text also changes the defaults for elements such as Compiler Error, Keyword, or Operator.

Automatic Items can inherit the background color from other display items such as **Plain Text**. Using this option, when you change the color of an inherited display item, the color of the related display items also change automatically. For example, if you selected the **Automatic** value for **Compiler Error** and later changed the color of **Plain Text** to Red, **Compiler Error** would also automatically inherit the color Red.

Default The color that appears for the item the first time you start Visual Studio. Clicking the **Use Defaults** button resets to this color.

Custom

Displays the Color dialog box to allow you to set a custom color for the item selected in the Display items list.

Bold

Select this option to display the text of selected **Display items** in bold text. Bold text is easier to identify in the editor.

Sample

Displays a sample of the font style, size, and color scheme for the **Show settings for** and **Display items** selected. You can use this box to preview the results as you experiment with different formatting options.

See Also

[Environment Options Dialog Box](#)

[Options Dialog Box](#)

[How to: Change Fonts and Colors](#)

Import and Export Settings, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Use this page to specify where your user settings file is saved and whether to use a team settings file. For more information about settings, see [Personalize the Visual Studio IDE](#).

See Also

[Environment Options Dialog Box](#)

International Settings, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

The International Settings page allows you to change the default language when you have more than one language version of the integrated development environment (IDE) installed on your machine. You can access this dialog box by selecting **Options** from the **Tools** menu and then choosing **International Settings** from the **Environment** folder. If this page does not appear in the list, select **Show all settings** in the **Options** dialog box.

NOTE

The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Language

Lists the available languages for the installed product language versions. This option is unavailable unless you have more than one language version installed on your machine. If multiple languages of products or a mixed language installation of products share the environment, the language selection is changed to **Same as Microsoft Windows**.

Caution

In a system with multiple languages installed, the Visual C++ build tools (cl.exe, link.exe, nmake.exe, bscmake.exe and related files) are not affected by this setting. These tools use the version for the last language installed. The build tools for the previously installed language are overwritten, because the Visual C++ build tools do not use the satellite DLL model.

See Also

[Install language packs](#)

[Environment Options Dialog Box](#)

Keyboard, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Use this page to set keyboard mappings. For more information about mappings, see [Identifying and Customizing Keyboard Shortcuts](#).

See Also

[Environment Options Dialog Box](#)

Notifications, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Use this option to stop ignoring any notifications that you previously chose to ignore through the Notifications dialog. For more information, see [Notifications](#).

See Also

[Environment Options Dialog Box](#)

Quick Launch, Environment, Options Dialog Box

12/22/2017 • 2 min to read • [Edit Online](#)

You can use **Quick Launch** to quickly search and execute actions for IDE assets such as options, templates, menus. You can't use **Quick Launch** to search for code and symbols. The **Quick Launch** search box is located at the top-right corner of the menu bar and is accessible by choosing the Ctrl+Q keys. Simply enter your search string in the box. To search for strings that contain @, use '@@'.

Quick Launch is enabled by default when you install Visual Studio. On the menu bar, you can show or hide **Quick Launch** by choosing **Tools**, **Options**. Expand the **Environments** node, and then choose **Quick Launch**. Select or clear the **Enable Quick Launch** check box. You can also enable or disable search categories on this page.

Category List

Quick Launch search results appear in four categories: **Most Recently Used**, **Menus**, **Options**, and **Open Documents**, along with the number of items in the category. To traverse through search results by category, choose the Ctrl+Q keys to show all the results from the next category. After the last category appears, Ctrl+Q shows you a few results from each category. You can use Ctrl+Shift+Q to navigate through the categories in reverse order. To view all the search results under a category, choose the category name.

You can use the following shortcuts to limit your search to specific categories.

| CATEGORY | SHORTCUT | SHORTCUT DESCRIPTION |
|--------------------|---|---|
| Most recently used | @mru For example, <code>@mru font</code> | Displays up to five of the items that you Most Recently Used . |
| Menus | @menu For example, <code>@menu font</code> | Limits the search to menu items. |
| Options | @opt For example, <code>@opt font</code> | Limits the search to settings in the Options dialog box. |
| Documents | @doc For example, <code>@doc font</code> | Limits the search to file names and paths of open documents for the search criteria, but doesn't search the text inside the files themselves. |

NOTE

You can change the shortcut keys on the **General**, **Keyboard** page in the **Options** dialog box.

Show Previous Results

By default, the search term that you enter is not persisted between search sessions. The search string is cleared if you search for a term, move the cursor outside the **Quick Launch** area, and then go back. To retain the search results, go to the **Options** dialog box, choose **Quick Launch**, and then select the **Show search results from previous search when Quick Launch is activated** check box. The next time you do a search, leave the Quick

Launch area, and come back, Quick Launch will retain the search term last used and also show you the search results.

For the most recent tips and tricks for using **Quick Launch**, see [The Visual Studio Blog](#).

See Also

[General User Interface Elements \(Visual Studio\)](#)

[Environment Options Dialog Box](#)

Startup, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Use this page to customize the Visual Studio start page or set a different default action when Visual Studio starts up. For more information, see [Customizing the Start Page](#).

See Also

[Environment Options Dialog Box](#)

Synchronized Settings, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Use this page to specify whether to synchronize (roam) your settings across multiple machines. For more information, see [Synchronized Settings](#).

See Also

[Environment Options Dialog Box](#)

Tabs and Windows, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Use this page to set options for how tabbed windows behave in the editor pane, and how tool windows behave. For more information, see [Customize window layouts in Visual Studio](#)

See Also

[Environment Options Dialog Box](#)

Task List, Environment, Options Dialog Box

1/26/2018 • 1 min to read • [Edit Online](#)

This Options page allows you to add, delete, and change the comment tokens that generate **Task List** reminders. To display these settings, select **Options** from the **Tools** menu, expand the **Environment** folder, and choose **Task List**.

Task List options

Confirm deletion of tasks

When selected, a message box is displayed whenever a User Task is deleted from the **Task List**, allowing you to confirm the deletion. This option is selected by default.

NOTE

To delete a Task Comment, use the link to find the comment, and then remove it from your code.

Show file names only

When selected, the **File** column of the **Task List** displays only the names of files to be edited, not their full paths.

Tokens

When you insert a comment into your code whose text begins with a token from the **Token List**, the **Task List** displays your comment as new entry whenever the file is opened for editing. You can click this **Task List** entry to jump directly to the comment line in your code. For more information, see [Using the Task List](#).

Token List

Displays a list of tokens, and allows you to add or remove custom tokens. Comment tokens are case sensitive in C# and Visual C++, but not in Visual Basic.

NOTE

If you do not type the desired token exactly as it appears in the **Token List**, a comment task will not be displayed in the **Task List**.

Priority

Sets the priority of tasks that use the selected token. Task comments that begin with this token are automatically assigned the designated priority in the **Task List**.

Name

Enter the token string. This enables the **Add** button. On **Add**, this string is included in the **Token List**, and comments that begin with this name will be displayed in the **Task List**.

Add

Enabled when you enter a new **Name**. Click to add a new token string using the values entered in the **Name** and **Priority** fields.

Delete

Click to delete the selected token from the **Token List**. You cannot delete the default comment token.

Change

Click to make changes to an existing token using the values entered in the **Name** and **Priority** fields.

NOTE

You cannot rename or delete the default comment token, but you can change its priority level.

See Also

[Using the Task List](#)

[Setting Bookmarks in Code](#)

[Environment Options Dialog Box](#)

Web Browser, Environment, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Sets options for both the internal Web browser and Internet Explorer. To access this dialog box, click **Options** on the **Tools** menu, expand the **Environment** folder, and select **Web Browser**.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose Import and Export Settings on the Tools menu. For more information, see [Personalize the Visual Studio IDE](#).

IMPORTANT

Opening certain files or components from the Web can execute code on your computer.

Home page

Sets the page displayed when you open the IDE Web Browser.

Search page

Lets you designate a Search page for the internal Web browser. This location can differ from the search page used by instances of Internet Explorer initiated outside of the integrated development environment (IDE).

View source in

Sets the editor used to open a Web page when you choose **View Source** on the page from the internal Web browser.

- **Source editor** Select to view source in the [editor](#).
- **HTML editor** Select to view source in the [HTML designer](#). Use this selection to edit the Web page in one of two views: Design view or the standard text-based Source view.
- **External editor** Select to view source in another editor. Specify the path of any editor you choose, for example, Notepad.exe.

Internet Explorer Options

Click to change options for Internet Explorer in the **Internet Properties** dialog box. Changes made in this dialog box affect both the internal Web browser and instances of Internet Explorer initiated outside of the Visual Studio IDE (for example, from the Start menu).

NOTE

Use the **Browse With** dialog box to replace the Visual Studio internal Web browser with a browser of your choice. You can access the Browse With dialog box from the right-click or context menu of, for example, an HTML file in your project.

See also

[Environment Options Dialog Box](#)
[General, Environment, Options Dialog Box](#)
[HTML Designer](#)

Text editor Options dialog box

2/11/2018 • 1 min to read • [Edit Online](#)

The editor options in the **Options** dialog box provide ways to customize the appearance and behavior of the editor.

See also

[Writing Code](#) describes different ways to use the code editor.

Options, Text Editor, General

2/11/2018 • 2 min to read • [Edit Online](#)

This dialog box lets you change global settings for the Visual Studio Code and Text Editor. To display this dialog box, click **Options** on the **Tools** menu, expand the **Text Editor** folder, and then click **General**.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Settings

Drag and drop text editing

When selected, enables you to move text by selecting it and dragging it with the mouse to another location within the current document or any other open document.

Automatic delimiter highlighting

When selected, delimiter characters that separate parameters or item-value pairs, as well as matching braces, are highlighted.

Track changes

When the code editor is selected, a vertical yellow line appears in the selection margin to mark code that has changed since the file was most recently saved. When you save the changes, the vertical lines become green.

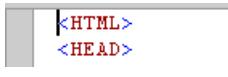
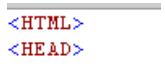
Auto-detect UTF-8 encoding without signature

By default, the editor detects encoding by searching for byte order marks or charset tags. If neither is found in the current document, the code editor attempts to auto-detect UTF-8 encoding by scanning byte sequences. To disable the auto-detection of encoding, clear this option.

Display

Selection margin

When selected, displays a vertical margin along the left edge of the editor's text area. You can click this margin to select an entire line of text, or click and drag to select consecutive lines of text.

| SELECTION MARGIN ON | SELECTION MARGIN OFF |
|---|---|
|  |  |

Indicator margin

When selected, displays a vertical margin outside the left edge of the editor's text area. When you click in this margin, an icon and ToolTip that are related to the text appear. For example, breakpoint or task list shortcuts appear in the indicator margin. Indicator Margin information does not print.

Vertical scroll bar

When selected, displays a vertical scrollbar which allows you to scroll up and down to view elements that fall outside the viewing area of the Editor. If vertical scrollbars are not available, you can use the Page Up, Page Down and cursor keys to scroll.

Horizontal scroll bar

When selected, displays a horizontal scrollbar which allows you to scroll from side-to-side to view elements that fall outside the viewing area of the Editor. If horizontal scrollbars are unavailable, you can use the cursor keys to scroll.

Highlight current line

When selected, displays a gray box around the line of code in which the cursor is located.

See Also

[Options, Text Editor, All Languages](#)

[Options, Text Editor, All Languages, Tabs](#)

[Options, Text Editor, File Extension](#)

[Identifying and Customizing Keyboard Shortcuts](#)

[Customizing the Editor](#)

[Using IntelliSense](#)

Options, Text Editor, File Extension

12/22/2017 • 1 min to read • [Edit Online](#)

This Options dialog allows you to specify how all files with certain file extensions will be handled by the Visual Studio integrated development environment (IDE). For each **Extension** that you enter, you can select an Editing Experience. This allows you to choose the IDE editor or designer in which documents of a certain type will open. To display these options, choose **Options** from the **Tools** menu, expand the **Text Editor** node, and select **File Extension**.

When you select an option "with Encoding," a dialog will be displayed whenever you open a document of that type that allows you to select an encoding scheme for that document. This can be helpful if you are preparing versions of your project documents for use on different platforms or in different target languages.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

UIElement List

Extension

Type the file extension whose Editing Experience in the IDE you wish to define.

Editor

Select the IDE editor or designer in which documents with this file extension will open. When you select an option "with Encoding," a dialog will be displayed whenever you open such a document that allows you to select an encoding scheme.

Add

Adds an entry that includes the specified **Extension** and **Editing Experience** to the Extension List.

Remove

Deletes the selected entry from the Extension List.

Extension List

Lists all extensions for which an Editing Experience has been specified.

Map extensionless files to

Select this option if you wish to specify how files without an extension will be handled by the IDE.

Extensionless file options

Provides the same list as **Editor**. Select the IDE editor or designer in which documents without file extensions will open.

See Also

[How to: Manage Editor Modes](#)

Options, Text Editor, All Languages

12/22/2017 • 3 min to read • [Edit Online](#)

This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To open this dialog box, select **Options** from the **Tools** menu. Within the **Text Editor** folder, expand the **All Languages** subfolder and then choose **General**.

Caution

This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the General options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the subfolder for that language and select its option pages.

A grayed checkmark is displayed when an option has been selected on the General options pages for some programming languages, but not for others.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Statement Completion

Auto list members

When selected, pop-up lists of available members, properties, values, or methods are displayed by IntelliSense as you type in the editor. Choose any item from the pop-up list to insert the item into your code. Selecting this option enables the **Hide advanced members** option.

Hide advanced members

When selected, shortens pop-up statement completion lists by displaying only those items most commonly used. Other items are filtered from the list.

Parameter information

When selected, the complete syntax for the current declaration or procedure is displayed under the insertion point in the editor, with all of its available parameters. The next parameter you can assign is displayed in bold.

Settings

Enable virtual space

When this option is selected and **Word wrap** is cleared, you can click anywhere beyond the end of a line in the Code Editor and type. This feature can be used to position comments at a consistent point next to your code.

Word wrap

When selected, any portion of a line that extends horizontally beyond the viewable editor area is automatically displayed on the next line. Selecting this option enables the **Show visual glyphs for word wrap** option.

NOTE

The **Virtual Space** feature is turned off while **Word Wrap** is on.

Show visual glyphs for word wrap

When selected, a return-arrow indicator is displayed where a long line wraps onto a second line.



Clear this option if you prefer not to display these indicators.

NOTE

These reminder arrows are not added to your code, and do not print. They are for reference only.

Apply Cut or Copy commands to blank lines when there is no selection

This option sets the behavior of the editor when you place the insertion point on a blank line, select nothing, and then Copy or Cut.

- When this option is selected, the blank line is copied or cut. If you then Paste, a new, blank line is inserted.
- When this option is cleared, the Cut command removes blank lines. However, the data on the Clipboard is preserved. Therefore, if you then use the Paste command, the content most recently copied onto the Clipboard is pasted. If nothing has been copied previously, nothing is pasted.

This setting has no effect on Copy or Cut when a line is not blank. If nothing is selected, the entire line is copied or cut. If you then Paste, the text of the entire line and its endline character are pasted.

TIP

To display indicators for spaces, tabs, and line ends, and thus distinguish indented lines from lines that are entirely blank, select **Advanced** from the **Edit** menu and choose **View White Space**.

Display

Line numbers

When selected, a line number appears next to each line of code.

NOTE

These line numbers are not added to your code, and do not print. They are for reference only.

Enable single-click URL navigation

When selected, the mouse cursor changes to a pointing hand as it passes over a URL in the editor. You can click the URL to display the indicated page in your Web browser.

Navigation bar

When selected, displays the **Navigation bar** at the top of the code editor. Its dropdown **Objects** and **Members** lists allow you to choose a particular object in your code, select from its members, and navigates to the declaration of the selected member in the Code Editor.

See Also

[Options, Text Editor, All Languages, Tabs](#)

[General, Environment, Options Dialog Box](#)

[Using IntelliSense](#)

Options, Text Editor, All Languages, Tabs

12/22/2017 • 2 min to read • [Edit Online](#)

This dialog box allows you to change the default behavior of the Code Editor. These settings also apply to other editors based upon the Code Editor, such as the HTML Designer's Source view. To display these options, select **Options** from the **Tools** menu. Within the **Text Editor** folder expand the **All Languages** subfolder, and then choose **Tabs**.

Caution

This page sets default options for all development languages. Remember that resetting an option in this dialog will reset the Tabs options in all languages to whatever choices are selected here. To change Text Editor options for just one language, expand the subfolder for that language and select its option pages.

If different settings are selected on the Tabs options pages for particular programming languages, then the message "The indentation settings for individual text formats conflict with each other," is displayed for differing **Indenting** options; and the message "The tab settings for individual text formats conflict with each other," is displayed for differing **Tab** options. For example, this reminder is displayed if the **Smart indenting** option is selected for Visual Basic, but **Block indenting** is selected for Visual C++.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Indenting

None

When selected, new lines are not indented. The insertion point is placed in the first column of a new line.

Block

When selected, new lines are automatically indented. The insertion point is placed at the same starting point as the preceding line.

Smart

When selected, new lines are positioned to fit the code context, per other code formatting settings and IntelliSense conventions for your development language. This option is not available for all development languages.

For example, lines enclosed between an opening brace ({) and a closing brace (}) might automatically be indented an extra tab stop from the position of the aligned braces.

Tabs

Tab size

Sets the distance in spaces between tab stops. The default is four spaces.

Indent size

Sets the size in spaces of an automatic indentation. The default is four spaces. Tab characters, space characters, or both will be inserted to fill the specified size.

Insert spaces

When selected, indent operations insert only space characters, not TAB characters. If the **Indent size** is set to 5, for

example, then five space characters are inserted whenever you press the TAB key or the **Increase Indent** button on the **Formatting** toolbar.

Keep tabs

When selected, indent operations insert as many TAB characters as possible. Each TAB character fills the number of spaces specified in **Tab size**. If the **Indent size** is not an even multiple of the **Tab size**, space characters are added to fill in the difference.

See Also

[Options, Text Editor, All Languages](#)

[General, Environment, Options Dialog Box](#)

Options, Text Editor, Basic (Visual Basic)

12/22/2017 • 2 min to read • [Edit Online](#)

The **VB Specific** property page, in the **Basic** folder of the **Text Editor** folder of the **Options (Tools menu)** dialog box contains the following properties:

Automatic insertion of end constructs

When you type—for example, the first line of a procedure declaration, `Sub Main` and press ENTER, the text editor adds a matching `End Sub` line. Similarly, if you add a `For` loop, the text editor adds a matching `Next` statement. When this option is selected, the code editor automatically adds the end construct.

Pretty Listing (reformatting) of code

The text editor reformats your code as appropriate. When this option is selected, the code editor will:

- Align your code to the correct tab position
- Recase keywords, variables, and objects to the correct case
- Add a missing `Then` to an `If...Then` statement
- Add parenthesis to function calls
- Add missing end quotes to strings
- Reformat exponential notation
- Reformat dates

Enable outlining mode

When you open a file in the code editor, you can view the document in outlining mode. See [Outlining](#) for more information. When this option is selected, the outlining feature is activated when you open a file.

Automatic insertion of Interface and MustOverride members

When you commit an `Implements` statement or an `Inherits` statement for a class, the text editor inserts prototypes for the members that have to be implemented or overridden, respectively.

Show procedure line separators

The text editor indicates visual scope of procedures. A line is drawn in the .vb source files of your project at locations listed in the following table:

| LOCATION IN .VB SOURCE FILE | EXAMPLE OF LINE LOCATION |
|---|---|
| After the close of a block declaration construct | <ul style="list-style-type: none">- At the end of a class, structure, module, interface, or enum- After a property, function, or sub- Not between the get and set clauses in a property |
| After a set of single line constructs | <ul style="list-style-type: none">- After the import statements, before a type definition in a class file- After variables declared in a class, before any procedures |
| After single line declarations (non-block level declarations) | <ul style="list-style-type: none">- Following import statements, inherits statements, variable declarations, event declarations, delegate declarations, and DLL declare statements |

Enable error correction suggestions

The text editor can suggest solutions to common errors and allow you to select the appropriate correction, which is then applied to your code.

Enable highlighting of references and keywords

The text editor can highlight all instances of a symbol or all of the keywords in a clause such as `If..Then`, `While...End While`, or `Try...Catch...Finally`. You can navigate between highlighted references or keywords by pressing CTRL+SHIFT+DOWN ARROW or CTRL+SHIFT+UP ARROW.

See Also

[General, Environment, Options Dialog Box](#)

[Options, Text Editor, All Languages, Tabs](#)

Options, Text Editor, C/C++, Formatting

12/22/2017 • 1 min to read • [Edit Online](#)

Lets you change the default behavior of the Code Editor when you are programming in C or C++.

To access this page, in the **Options** dialog box, in the left pane, expand **Text Editor**, expand **C/C++**, and then click **Formatting**.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalize the Visual Studio IDE](#).

C/C++ Options

Enable automatic Quick Info ToolTips

Enables or disables the Quick Info IntelliSense feature.

Inactive Code

Show Inactive Code Blocks

Code that is inactive due to `#ifdef` declarations are colorized differently to help you identify it.

Disable Inactive Code Opacity

Inactive code can be identified by using color instead of transparency.

Inactive Code Opacity Percent

The degree of opacity for inactive code blocks can be customized.

Indentation

Indent Braces

You can configure how braces are aligned when you press ENTER after you begin a code block, for example, a function or a `for` loop. The braces can either be aligned with the first character of the code block or indented.

Automatic Indentation On Tab

You can configure what happens on the current code line when you press TAB. Either the line is indented or a tab is inserted.

Miscellaneous

Enumerate the comments in the Task List window

The editor can scan open source files for preset words in the comments. It creates an entry in the **Task List** window for any keywords that it finds.

Highlight Matching Tokens

When the cursor is next to a brace, the editor can highlight the matching brace so that you can more easily see the contained code.

Outlining

Enter outlining mode when files open

When you bring a file into the text editor, you can enable the outlining feature. For more information, see [Outlining](#).

When this option is selected, the outlining feature is enabled when you open a file.

Automatic outlining of #pragma region blocks

When this option is selected, automatic outlining for [pragma directives](#) is enabled. This lets you expand or collapse pragma region blocks in outlining mode.

Automatic outlining of statement blocks

When this option is selected, automatic outlining is enabled for the following statement constructs:

- [if-else](#)
- [switch Statement \(C++\)](#)
- [while Statement \(C++\)](#)

See Also

[General, Environment, Options Dialog Box](#)

[Using IntelliSense](#)

Options, Text Editor, C/C++, Advanced

12/22/2017 • 7 min to read • [Edit Online](#)

By changing these options, you can change the behavior related to IntelliSense and the browsing database when you're programming in C or C++.

To access this page, in the **Options** dialog box, in the left pane, expand **Text Editor**, expand **C/C++**, and then choose **Advanced**.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. See [Personalize the Visual Studio IDE](#).

Browsing/Navigation

You should never choose these options except in the rare case where a solution is so large that the database activity consumes an unacceptable amount of system resources.

Disable Database

All use of the code browsing database (SDF), all other Browsing/Navigation options, and all IntelliSense features except for #include Auto Complete are disabled.

Disable Database Updates

The database will be opened read-only, and no updates will be performed as files are edited. Most features will still work. However, as edits are made, the data will become stale, and you'll get incorrect results.

Disable Database Auto Updates

The code browsing database won't be automatically updated when source files are modified. However, if you open **Solution Explorer**, open the shortcut menu for the project, and then choose **Rescan Solution**, all out-of-date files will be checked, and the database will be updated.

Disable Implicit Files

The code browsing database doesn't collect data for files that aren't specified in a project. A project contains source files and header files that are explicitly specified. Implicit files are included by explicit files (for example, afxwin.h, windows.h, and atlbase.h). Normally, the system finds these files and also indexes them for various browsing features (including Navigate To). If you choose this option, those files aren't indexed, and some features aren't available for them. If you choose this option, "Disable Implicit Cleanup" and "Disable External Dependencies" are also implicitly chosen.

Disable Implicit Cleanup

The code browsing database doesn't clean up implicit files that are no longer referenced. This option prevents implicit files from being removed from the database when they're no longer used. For example, if you add an `#include` directive that references mapi.h to one of your source files, mapi.h will be found and indexed. If you then remove the `#include` and the file isn't referenced elsewhere, information about it will eventually be removed unless you choose this option. (See the **Rescan Solution Interval** option.) This option is ignored when you explicitly rescan the solution.

Disable External Dependencies Folders

The External Dependencies folder for each project isn't created or updated. In **Solution Explorer**, each project

contains an External Dependencies folder, which contains all implicit files for that project. If you choose this option, that folder doesn't appear.

Recreate Database

Recreate the code browsing database from nothing the next time that the solution loads. If you choose this option, the SDF database file is deleted the next time you load the solution, thus causing the database to be recreated and all files indexed.

Rescan Solution Interval

A 'Rescan Solution Now' job is scheduled for the interval that you specify. You must specify between 0 and 5000 minutes. The default value is 60 minutes. While the solution is rescanned, file timestamps are checked to determine whether a file was changed outside of the IDE. (Changes that are made in the IDE are automatically tracked, and files are updated.) Implicitly included files are checked to determine whether they're all still referenced.

Diagnostic Logging

These options are provided in case Microsoft asks you to collect advanced information to diagnose an issue. The logging information isn't useful for users, and we recommend that you leave it disabled.

Enable Logging

Enables diagnostic logging to the output window.

Logging Level

Set the log verbosity, from 0 to 5.

Logging Filter

Filters displayed event types by using a bitmask.

Set by using a sum of any of the following options:

- 0 - None
- 1 - General
- 2 - Idle
- 4 - WorkItem
- 8 - IntelliSense
- 16 - ACPerf
- 32 - ClassView

Fallback Location

The fallback location is where the SDF and IntelliSense support files (for example, iPCH) are put when the primary location (same directory as solution) isn't used. This situation could occur the user doesn't have the permissions to write to the solution directory or the solution directory is on a slow device. The default fallback location is in the user's temp directory.

Always Use Fallback Location

Indicates that the code browsing database and IntelliSense files should always be stored in a folder that you specify as your "Fallback Location", not next to the .sln file. The IDE will never try to put the SDF or iPCH files next to the solution directory and will always use the fallback location.

Do Not Warn If Fallback Location Used

You aren't informed or prompted if a 'Fallback Location' is used. Normally, the IDE will tell you if it had to use the fallback location. This option turns off that warning.

Fallback Location

This value is used as a secondary location to store the code browsing database or IntelliSense files. By default, your temporary directory is your fallback location. The IDE will create a subdirectory under the specified path (or the temp directory) that includes the name of the solution along with a hash of the full path to the solution, which avoids issues with solution names being identical.

IntelliSense

Auto Quick Info

Enables QuickInfo tooltips when you move the pointer over text.

Disable IntelliSense

Disables all IntelliSense features. The IDE does not create VCPkgSrv.exe processes to service IntelliSense requests, and no IntelliSense features will work (QuickInfo, Member List, Auto Complete, Param Help). Semantic colorization and reference highlighting are also disabled. This option doesn't disable browsing features that rely solely on the database (including Navigation Bar, ClassView, and Property window).

Disable Auto Updating

IntelliSense updating is delayed until an actual request for IntelliSense is made. This delay can result in a longer execution time of the first IntelliSense operation on a file, but it may be helpful to set this option on very slow or resource-constrained machines. If you choose this option, you also implicitly choose the "Disable Error Reporting" and "Disable Squiggles" options.

Disable Error Reporting

Disables reporting of IntelliSense errors through squiggles and the Error List window. Also disables the background parsing that's associated with error reporting. If you choose this option, you also implicitly choose the "Disable Squiggles" option.

Disable Squiggles

Disables IntelliSense error squiggles. The red "squiggles" don't show in the editor window, but the error will still appear in the Error List window.

Disable #include Auto Complete

Disables auto-completion of `#include` statements.

Use Forward Slash in #include Auto Complete

Triggers auto-completion of `#include` statements when "/" is used. The default delimiter is backslash "\". The compiler can accept either, so use this option to specify what your code base uses.

Max Cached Translation Units

The maximum number of translation units that will be kept active at any one time for IntelliSense requests. You must specify a value between 2 and 15. This number directly relates to the maximum number of VCPkgSrv.exe processes that will run (for a given instance of Visual Studio). The default value is 2, but if you have available memory, you can increase this value and possibly achieve slightly better performance on IntelliSense.

For more information about translation units, see [Phases of Translation](#).

Member List Dot-To-Arrow

Replaces '.' with '>' when applicable for Member List.

Disable Aggressive Member List

The member list doesn't appear while you type the name of a type or variable. The list appears only after you type one of the commit characters, as defined in the **Member List Commit Characters** option.

Disable Member List Keywords

Language keywords such as `void`, `class`, `switch` don't appear in member list suggestions.

Disable Member List Code Snippets

Code snippets don't appear in member list suggestions.

Disable Semantic Colorization

Turns off all code colorization except for language keywords, strings, and comments.

Smart Member List Commit

Adds a line when you choose the Enter key at the end of a fully typed word.

Member List Filter Mode

Sets the type of matching algorithm. **Fuzzy** finds the most possible matches because it uses an algorithm that's similar to a spell-checker to find matches that are similar but not identical. **Smart filtering** matches substrings even if they're not at the start of a word. **Prefix** only matches on identical substrings that start at the beginning of the word.

Member List Commit Characters

Specifies the characters that cause the currently highlighted Member List suggestion to be committed. You can add or remove characters from this list.

References

Disable Resolving

For performance reasons, 'Find All References' displays raw textual search results by default instead of using IntelliSense to verify each candidate. You can clear this check box for more accurate results on all find operations. To filter on a per-search basis, open the shortcut menu for the result list, and then choose "Resolve Results."

Hide Unconfirmed

Hide unconfirmed items in the 'Find All References' results. If you unset the "Disable Resolving" option, you can use this option to hide unconfirmed items in the results.

Disable Reference Highlighting

Text Editor

Enable Expand Scopes

If enabled, you can surround selected text with curly braces by typing '{' into the text editor.

Enable Expand Precedence

If enabled, you can surround selected text with parentheses by typing '(' into the text editor.

See Also

[Setting Language-Specific Editor Options](#)

Options, Text Editor, C/C++, Experimental

3/12/2018 • 1 min to read • [Edit Online](#)

By changing these options, you can change the behavior related to IntelliSense and the browsing database when you're programming in C or C++. These features are truly experimental, and may be modified or removed from Visual Studio in a future release. This topic describes the options in Visual Studio 2017. For Visual Studio 2015, see [Options, Text Editor, C/C++, Experimental](#)

To access this property page, press **Control + Q** to activate **Quick Launch** and then type "experimental". Quick Launch will find the page after the first few letters. You can also get to it by choosing **Tools | Options** and expanding **Text Editor**, then **C/C++**, and then choosing **Experimental**.

These features are available in a Visual Studio 2017 installation.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. See [Personalize the Visual Studio IDE](#).

Enable Predictive IntelliSense

Predictive IntelliSense limits the number of results displayed in the IntelliSense dropdown list so that you see only results that are relevant in the context. For example, if you type `int x =` and invoke the IntelliSense dropdown, you will see only integers or functions that return integers. Predictive IntelliSense is turned off by default.

Enable Faster Project Load

Visual Studio 2017 version 15.3 and later: This feature is now called **Enable Project Caching** and has moved to the [VC++ Project Settings](#) property page. This option enables Visual Studio to cache project data so that when you open the project the next time, it can load that cached data rather than re-computing it from the project files. Using cached data can speed up the project load time significantly.

Additional Features in the Visual Studio Marketplace

You can browse additional text editor features in the [Visual Studio Marketplace](#). An example is [C++ Quick Fixes](#), which supports the following:

- **Add missing #include** - Suggests relevant #include's for unknown symbols in your code
- **Add using namespace/Fully qualify symbol** - Like the previous item, but for namespaces
- **Add missing semicolon**
- **Online help** - Search online help for your error messages
- And more...

See also

- [Setting Language-Specific Editor Options](#)
- [Refactoring in C++ \(VC Blog\)](#)

Options, Text Editor, C#, Formatting

2/11/2018 • 1 min to read • [Edit Online](#)

Use the **Formatting** options page to set options for formatting code in the Code Editor. To access this options page, choose **Tools > Options**, and then choose **Text Editor > C# > Code Style > Formatting**.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

General settings

The General settings affect how the Code Editor applies formatting options to code.

UIElement list

| LABEL | DESCRIPTION |
|--|--|
| Automatically format when typing | When deselected, the format statement on ; and format block on } options are disabled. |
| Automatically format statement on ; | When selected, formats statements at completion according to the formatting options selected for the editor. |
| Automatically format block on } | When selected, formats code blocks according to the formatting options selected for the editor as soon as you complete the code block. |
| Automatically format on return | When selected, formats text when Enter is pressed, to fit the formatting options selected for the editor. |
| Automatically format on paste | When selected, formats text that is pasted into the editor to fit the formatting options selected for the editor. |

Preview window

The **Indentation**, **New Lines**, **Spacing**, and **Wrapping** options panes each display a preview window. The preview window shows the effect of each option. To use the preview window, select a formatting option. The preview window shows an example of the selected option. When you change a setting by selecting a radio button or check box, the preview window updates to show the effect of the new setting.

Remarks

Indentation options on the **Tabs** pages for each language only determine where the Code Editor places the cursor when you press **Enter** at the end of a line. Indentation options under **Formatting** apply when code is formatted automatically, for example, when you paste code into the file while **Automatically format on paste** is selected, and when the block being formatted is typed manually.

See also

[General, Environment, Options Dialog Box](#)

Options, Text Editor, C#, Advanced

2/11/2018 • 2 min to read • [Edit Online](#)

Use the **Advanced** options page to modify the settings for editor formatting, code refactoring, and XML documentation comments for C#. To access this options page, choose **Tools** > **Options**, and then choose **Text Editor** > **C#** > **Advanced**.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Analysis

- Enable full solution analysis
 - Enables code analysis on all files in the solution, not just open code files. For more information, see [Full solution analysis](#).
- Perform editor feature analysis in external process (experimental)

Using Directives

- Place 'System' directives first when sorting usings
- Separate using directive groups
- Suggest usings for types in reference assemblies
- Suggest usings for types in NuGet packages

Highlighting

- Highlight references to symbol under cursor

When the cursor is positioned inside a symbol, or when you click a symbol, all the instances of that symbol in the code file are highlighted.

- Highlight related keywords under cursor

Outlining

- Enter outlining mode when files open

When selected, automatically outlines the code file, which creates collapsible blocks of code. The first time a file is opened, #regions blocks and inactive code blocks collapse.

- Show procedure line separators
- Show outlining for declaration level constructs
- Show outlining for code level constructs

- Show outlining for comments and preprocessor regions
- Collapse #regions when collapsing to definitions

Fading

- Fade out unused usings
- Fade out unreachable code

Block Structure Guides

- Show guides for declaration level constructs
- Show guides for code level constructs

Editor Help

- Generate XML documentation comments for `///`

When selected, inserts the XML elements for XML documentation comments after you type the `///` comment introduction. For more information about XML documentation, see [XML Documentation Comments \(C# Programming Guide\)](#).

- Insert * at the start of new lines when writing `/* */` comments
- Show preview for rename tracking
- Split string literals on enter
- Report invalid placeholders in 'string.Format' calls

Extract Method

- Don't put ref or out on custom struct

Implement Interface or Abstract Class

- When inserting properties, events and methods, place them with other members of the same kind, or at the end
- When generating properties, prefer throwing properties or prefer auto properties

See also

- [How to: Insert XML comments for documentation generation](#)
[XML Documentation Comments \(C# Programming Guide\)](#)
[Documenting your code with XML comments \(C# Guide\)](#)
[Setting language-specific editor options](#)
[C# IntelliSense](#)

Options, Text Editor, C#, IntelliSense

2/11/2018 • 2 min to read • [Edit Online](#)

Use the **IntelliSense** options page to modify settings that affect the behavior of IntelliSense for C#. To access this options page, choose **Tools > Options**, and then choose **Text Editor > C# > IntelliSense**.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

The **IntelliSense** options page contains the following options:

Completion Lists

- Show completion list after a character is typed*

When this option is selected, IntelliSense automatically displays the completion list when you begin typing.

When this option is not selected, IntelliSense completion is still available from the **IntelliSense** menu or by pressing **Ctrl+Space**.

- Show completion list after a character is deleted
- Highlight matching portions of completion list items
- Show completion list filters
- Show name suggestions

Snippets behavior

- Never include snippets

When this option is selected, IntelliSense never adds aliases for C# code snippets to the completion list.

- Always include snippets

When this option is selected, IntelliSense adds aliases for C# code snippets to the completion list. In the case where the code snippet alias is the same as a keyword, for example, `class`, the keyword is replaced by the shortcut. For more information, see [C# Code Snippets](#).

- Include snippets when ?-Tab is typed after an identifier

When this option is selected, IntelliSense adds aliases for C# code snippets to the completion list when **?+Tab** is pressed after an identifier

Enter key behavior

- Never add new line on enter

Specifies that a new line is never added automatically after selecting an item in the completion list and pressing **Enter**.

- Only add new line on enter after end of fully typed word

Specifies that if you type all the characters for an entry in the completion list and then press **Enter**, a new

line is added automatically and the cursor moves to the new line.

For example, if you type `else` and then press **Enter**, the following appears in the editor:

```
else
```

```
| (cursor location)
```

However, if you type only `e1` and then press **Enter**, the following appears in the editor:

```
else| (cursor location)
```

- Always add new line on enter

Specifies that if you type *any* of the characters for an entry in the completion list and then press **Enter**, a new line is added automatically and the cursor moves to the new line.

See also

[General, Environment, Options Dialog Box](#)

[Using IntelliSense](#)

Options, Text Editor, JavaScript, Formatting

12/22/2017 • 2 min to read • [Edit Online](#)

Use the **Formatting** page of the **Options** dialog box to set options for formatting code in the Code Editor. To access this page, on the menu bar, choose **Tools**, **Options**, and then expand **Text Editor**, **JavaScript**, and **Formatting**.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

Automatic Formatting

These options determine when formatting occurs in **Source** view.

UIElement List

| OPTION | DESCRIPTION |
|--|--|
| Format completed line on Enter | When this option is selected, the Code Editor automatically formats the line when you choose the Enter key. |
| Format completed statement on ; | When this option is selected, the Code Editor automatically formats the line when you choose the semicolon key. |
| Format completed block on } | When this option is selected, the Code Editor automatically formats the line when you choose the closing brace key. |
| Format on paste | When this option is selected, the Code Editor reformats code when you paste it into the editor. The editor uses the currently defined formatting rules. If this option is not selected, the editor uses the original formatting of the pasted-in code. |

New Lines

These options determine whether the Code Editor puts an open brace for functions and control blocks on a new line.

UIElement List

| OPTION | DESCRIPTION |
|---|--|
| Place open brace on new line for functions | When this option is selected, the Code Editor moves the open brace associated with a function to a new line. |

| OPTION | DESCRIPTION |
|--|--|
| Place open brace on new line for control blocks | When this option is selected, the Code Editor moves the open brace associated with a control block (for example, <code>if</code> and <code>while</code> control blocks) to a new line. |

Spacing

These options determine how spaces are inserted in **Sourceview**.

UIElement List

| OPTION | DESCRIPTION |
|--|---|
| Insert space after comma delimiter | When this option is selected, the Code Editor adds a space after comma delimiters. |
| Insert space after semicolon in 'for' statement | When this option is selected, the Code Editor adds a space after each semicolon in the first line of a <code>for</code> loop. |
| Insert space before and after binary operators | When this option is selected, the Code Editor adds a space before and after binary operators (for example, <code>+</code> , <code>-</code> , <code>&&</code> , <code> </code>). |
| Insert space after keywords in control flow statements | When this option is selected, the Code Editor adds a space after JavaScript keywords in control flow statements. |
| Insert space after function keyword for anonymous functions. | When this option is selected, the Code Editor adds a space after the <code>function</code> keyword for anonymous functions. |
| Insert space after opening and before closing non-empty parenthesis | When this option is selected, the Code Editor adds a space after the opening parenthesis and before the closing parenthesis if non-empty characters are present within the parentheses. |

See Also

[General, Environment, Options Dialog Box](#)

Options, Text Editor, JavaScript, IntelliSense

12/22/2017 • 3 min to read • [Edit Online](#)

Use the **IntelliSense** page of the **Options** dialog box to modify settings that affect the behavior of IntelliSense for JavaScript. You can access the **IntelliSense** page by choosing **Tools**, **Options** on the menu bar, and then expanding **Text Editor**, **JavaScript**, **IntelliSense**.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

The **IntelliSense** page contains the following sections:

Validation

You can use these options to set preferences for how the JavaScript editor validates syntax in your document.

UIElement List

Show syntax errors

When this check box is not selected, the JavaScript code editor does not show syntax errors. This is useful if you are working with code that you didn't write and you don't intend to fix syntax errors.

When this check box is selected, you have the option to select the **Show errors as warning** check box.

Show errors as warnings

When this check box is selected, JavaScript errors are shown as warnings instead of errors in the error list.

Download remote references (e.g. `http://`) for files in the miscellaneous files project

When this check box is selected, and if you have a JavaScript file opened outside the context of a project, Visual Studio will download remote JavaScript files referenced in the file for the purpose of providing IntelliSense information. If this option is selected, files will download when you include them as a reference in your JavaScript file.

NOTE

For Web projects, remote files referenced in your project are downloaded by default.

Statement Completion

You can use these options to change the behavior of IntelliSense statement completion.

UIElement List

Only use tab or enter to commit

When this check box is selected, the JavaScript code editor appends statements with items selected in the completion list only after you choose the Tab or Enter key. When this check box is not selected, other characters, such as a period, comma, colon, open parenthesis, and open brace ({), can also append statements with the selected

items.

References

You can use these options to specify the types of IntelliSense .js files that are in scope for different JavaScript project types. The IntelliSense references are typically used to provide IntelliSense support for global objects. You can also use this page to set the loading order for scripts that must be loaded at run time, and to add IntelliSense extension files.

UIElement List

Reference groups

This option specifies the reference group type. Three reference groups are supported:

You can use pre-defined reference groups to specify that particular IntelliSense .js files are in scope for different JavaScript projects. Four reference groups are available:

- Implicit (*Windows version*), for Windows 8.x Store apps using JavaScript. Files included in this group are in scope for every .js file opened in the Code Editor for Windows 8.x Store apps using JavaScript.
- Implicit (Web), for HTML5 projects. Files included in this group are in scope for every .js file opened in the Code Editor for these project types.
- Dedicated worker reference groups, for HTML5 Web Workers. Files specified in this group are in scope for .js files that have an explicit reference to a dedicated worker reference group.
- Generic, for other JavaScript project types.

Included files

This option specifies the order in which files are loaded into the context of the language service. You can configure the order by using the **Remove**, **Move Up**, and **Move Down** buttons. For IntelliSense to work correctly, a file that is dependent on another must be loaded after the other file.

Caution

If an object is defined unconditionally in two or more implicit references, the last reference in this list will be used to define the object.

Add a reference to the group

This option provides a way to add additional IntelliSense .js files by browsing to the appropriate files.

See Also

[JavaScript IntelliSense](#)

Options, Text Editor, XAML, Formatting

1/26/2018 • 3 min to read • [Edit Online](#)

Use the **Formatting** property page to specify how elements and attributes are formatted in your XAML documents. To open the **Options** dialog box, click the **Tools** menu and then click **Options**. To access the **Formatting** property page, expand the **Text Editor, XAML, Formatting** node.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Auto-Formatting Events

Auto-formatting may occur when any of the following events is detected.

- Completion of an end tag or simple tag.
- Completion of a start tag.
- Pasting from the clipboard.
- Formatting keyboard commands.

You can specify which events cause auto-formatting.

| | |
|---|--|
| On completion of end tag or simple tag | Auto-formatting occurs when you finish typing an end tag or a simple tag. A simple tag has no attributes, for example <code><Button /></code> . |
| On completion of start tag | Auto-formatting occurs when you finish typing a start tag. |
| On paste from clipboard | Auto-formatting occurs when you paste XAML from the clipboard into XAML view. |

Quotation Mark Style

This setting indicates whether attribute values are enclosed in single or double quotation marks. The auto-formatter and IntelliSense auto-completion both use this setting.

Once you set this option, only attributes subsequently added either using the designer or manually in the XAML view are affected.

| | |
|---------------------------|---|
| Double quotes ("") | Attribute values are enclosed in double quotes. <code><Button Name="button1">Hello</Button></code> |
|---------------------------|---|

| | |
|---------------------------|--|
| Single quotes ('') | Attribute values are enclosed in single quotes. |
| | <pre><Button Name='button1'>Hello</Button></pre> |

Tag Wrapping

You can specify a line length for tag wrapping. When tag wrapping is enabled, any XAML subsequently added by using the designer will be wrapped appropriately.

| | |
|---|--|
| Wrap tags that exceed specified length | Specifies whether lines are wrapped at the line length specified by Length . |
| Length | The number of characters a line may contain. If necessary, some XAML lines might exceed the specified line length. |

Attribute Spacing

Use this setting to control how attributes are arranged in your XAML document

| | |
|---|---|
| Preserve newlines and spaces between attributes | New lines and spaces between attributes are not affected by auto-formatting. <pre><Button Height="23" Name="button1" Width="75">Hello</Button></pre> |
| Insert a single space between attributes | Attributes occupy one line, with one space separating adjacent attributes. Tag wrapping settings are applied. <pre><Button Height="23" Name="button1" Width="75">Hello</Button></pre> |
| Position each attribute on a separate line | Each attribute occupies its own line. This is useful when many attributes are present. <pre><Button Height="23" Name="button1" Width="75">Hello</Button></pre> |
| Position first attribute on same line as start tag | When checked, the first attribute appears on the same line as the element's start tag. <pre><Button Height="23" Name="button1" Width="75">Hello</Button></pre> |

Element Spacing

Use this setting to control how elements are arranged in your XAML document

| | |
|--|--|
| Preserve new lines in content | Empty lines in element content are not removed. <code><Grid></code> <code> </code> <code><Button Name="button1">Hello</Button></code> <code>` `</code> |
| Collapse multiple empty lines in content to a single line | Empty lines in element content are collapsed to a single line. <code><Grid></code> <code> `<Button Name="button1">Hello</Button>` </code> <code></Grid></code> |
| Remove empty lines in content | All empty lines in element content are removed. <code><Grid></code> <code><Button Name="button1">Hello</Button></code> <code></Grid></code> |

Miscellaneous section, Auto Insert

Use this setting to control when tags and quotes are automatically generated.

| | |
|--|---|
| Closing tags | Specifies whether an element's closing tag is automatically generated when you close the opening tag with the greater than character (>). |
| Attribute quotes | Specifies whether enclosing quotes are generated when an attribute value is selected from the statement completion drop-down list. |
| Closing braces for MarkupExtensions | Specifies whether a markup extension's closing brace () is automatically generated when you type the opening brace character (())�. |
| Commas to separate MarkupExtension parameters | Specifies whether commas are generated when you type more than one parameter in a markup extension. |

See also

[XAML in WPF](#)

Projects and Solutions, Options Dialog Box

12/22/2017 • 3 min to read • [Edit Online](#)

Sets Visual Studio behavior related to projects and solutions. To access these options, select **Tools > Options**, expand **Projects and Solutions**, and click **General**.

The default paths for project and template folders are set through the **Locations** tab in the same dialog box.

NOTE

The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. This Help page was written with the **General Development settings** in mind. To view or change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

General tab options

Always show Error List if build finishes with errors

Opens the **Error List** window on build completion, only if a project failed to build. Errors that occur during the build process are displayed. When this option is cleared, the errors still occur but the window does not open when the build is complete. This option is enabled by default.

Track Active Item in Solution Explorer

When selected, **Solution Explorer** automatically opens and the active item is selected. The selected item changes as you work with different files in a project or solution, or different components in a designer. When this option is cleared, the selection in **Solution Explorer** does not change automatically. This option is enabled by default.

Show advanced build configurations

When selected, the build configuration options appear on the **Project Property Pages** dialog box and the **Solution Property Pages** dialog box. When cleared, the build configuration options do not appear on the **Project Property Pages** dialog box and the **Solution Property Pages** dialog box for Visual Basic and Visual C# projects that contain one configuration or the two configurations debug and release. If a project has a user-defined configuration, the build configuration options are shown.

When unselected, the commands on the **Build** menu, such as **Build Solution**, **Rebuild Solution**, and **Clean Solution**, are performed on the Release configuration and the commands on the **Debug** menu, such as **Start Debugging** and **Start Without Debugging**, are performed on the Debug configuration.

Always show solution

When selected, the solution and all commands that act on solutions are always shown in the IDE. When cleared, all projects are created as stand-alone projects and you do not see the solution in Solution Explorer or commands that act on solutions in the IDE if the solution contains only one project.

Save new projects when created

When selected, you can specify a location for your project in the **New Project** dialog box. When cleared, all new projects are created as temporary projects. When you are working with temporary projects, you can create and experiment with a project without having to specify a disk location.

Warn user when the project location is not trusted

If you attempt to create a new project or open an existing project in a location that is not fully trusted (for example, on a UNC path or an HTTP path), a message is displayed. Use this option to specify whether the message is

displayed each time that you attempt to create or open a project in a location that is not fully trusted.

Show Output window when build starts

Automatically displays the Output Window in the IDE at the outset of solution builds. For more information, see [How to: Control the Output Window](#).

Prompt for symbolic renaming when renaming files

When selected, Visual Studio displays a message box asking whether or not it should also rename all references in the project to the code element.

Prompt before moving files to a new location

When selected, Visual Studio displays a confirmation message box before the locations of files are changed by actions in Solution Explorer.

Locations tab options

Projects location

Specifies the default location where Visual Studio creates new projects and solution folders. Several dialog boxes also use the location set in this option for folder starting points. For example, the Open Project dialog box uses this location for the My Projects shortcut.

User project templates location

Specifies the default location that the **New Project** dialog box uses to create the list of **My Templates**. For more information, see [How to: Locate and Organize Templates](#).

User item templates location

Specifies the default location that the **Add New Item** dialog box uses to create the list of **My Templates**. For more information, see [How to: Locate and Organize Templates](#).

See Also

- [Options Dialog Box, Projects and Solutions, Build and Run](#)
- [Options Dialog Box, Projects and Solutions, Web Projects](#)

Options Dialog Box, Projects and Solutions, Build and Run

1/26/2018 • 1 min to read • [Edit Online](#)

In this dialog box, you can specify the maximum number of Visual C++ or C# projects that can build at the same time, certain default build behaviors, and some build log settings. To access these options, select **Tools > Options** expand **Projects and Solutions**, and select **Build and Run**.

Maximum number of parallel project builds

Specifies the maximum number of Visual C++ and C# projects that can build at the same time. To optimize the build process, the maximum number of parallel project builds is automatically set to the number of CPUs of your computer. The maximum is 32.

Only build startup projects and dependencies on Run

Builds only the startup project and its dependencies when you use the F5 key, select the **Debug > Start** menu command, or applicable commands on the **Build** menu. If clear, all projects and dependencies are build.

On Run, when projects are out of date

Applies to Visual C++ projects only.

When running a project with F5 or the **Debug > Start** command, the default setting **Prompt to build** displays a message if a project configuration is out of date. Select **Always build** to build the project every time it is run. Select **Never build** to suppress all automatic builds when a project is run.

On Run, when build or deployment errors occur

Applies to Visual C++ projects only.

When running a project with F5 or the **Debug > Start** command, the default setting **Prompt to launch** displays a message if a project should be run even if the build failed. Select **Launch old version** to automatically launch the last good build, which could result in mismatches between the running code and the source code. Select **Do not launch** to suppress the message.

For new solutions use the currently selected project as the startup project

When this option is set, new solutions use the currently selected project as the startup project.

MSBuild project build output verbosity

Determines how much information appears in the **Output** window for the build.

MSBuild project build log file verbosity

Applies to Visual C++ projects only.

Determines how much information is written to the build log file, which is located at
`...\\ProjectName\\Debug\\ProjectName.log`.

See also

[Compiling and Building](#)

[Options Dialog Box, Projects and Solutions](#)

[Options Dialog Box, Projects and Solutions, Web Projects](#)

Options Dialog Box, Projects and Solutions, Web Projects

1/26/2018 • 1 min to read • [Edit Online](#)

Sets the Web server that Web projects will use for development within Visual Studio. To access these options, select **Tools > Options** expand **Projects and Solutions**, and select **Web Projects**.

By default, running a Web project in Visual Studio uses the Visual Studio Development Server. For more information, see [Web Servers in Visual Studio for ASP.NET Web Projects](#).

NOTE

The options available in dialog boxes, and the names and locations of menu commands you see, might differ from what is described in Help depending on your active settings or edition. This Help page was written with the **Web settings** in mind. To view or change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Settings

Use the 64-bit version of IIS Express for web sites and projects

Select this option to use IIS Express instead of the Visual Studio Development Server. For more information, see [Introducing IIS Express](#) and [IIS Express Overview](#).

Warn before running web applications when there are errors in the error list

If this option is set, you will be warned if you try to run your web application when it does not compile without errors.

See Also

[Options Dialog Box, Projects and Solutions](#)

[Options Dialog Box, Projects and Solutions, Build and Run](#)

Visual Basic Defaults, Projects, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

Specifies the default settings for Visual Basic project options. When a new project is created, the specified option statements will be added to the project header in the Code Editor. The options apply to all Visual Basic projects.

To access this dialog box, on the **Tools** menu, click **Options**, expand the **Projects and Solutions** folder, and then click **VB Defaults**.

Option Explicit

Sets the compiler default so that explicit declarations of variables are required. By default, **Option Explicit** is set to **On**. For more information, see [/optionexplicit](#).

Option Strict

Sets the compiler default so that explicit narrowing conversions are required and late binding is not allowed. By default, **Option Strict** is set to **Off**. For more information, see [/optionstrict](#).

Option Compare

Sets the compiler default for string comparisons: binary (case-sensitive) or text (case-insensitive.) By default, **Option Compare** is set to **Binary**. For more information, see [/optioncompare](#).

Option Infer

Sets the compiler default for local type inference. By default, **Option Infer** is set to **On** for newly created projects and to **Off** for migrated projects created in earlier versions of Visual Basic. For more information, see [/optioninfer](#).

See Also

[Solutions and Projects](#)

VC++ Project Settings, Projects and Solutions, Options Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

This dialog box lets you define Visual C++ build and project settings related to logging, performance, and supporting file types.

To access this dialog box

1. On the **Tools** menu, click **Options**.
2. Select **Projects and Solutions**, and then select **VC++ Project Settings**.

Build Logging

Yes

Turns on generation of the build log file. This option generates BuildLog.htm, which can be found in the project's intermediate files directory. Every fresh build overwrites the previous BuildLog.htm file.

No

Turns off generation of the build log file.

Show Environment in Log

Yes

Lists environment variables in the build log file. This option specifies to echo all environment variables, during builds of Visual C++ projects, into the build log file.

No

Exclude environment variables from the build log file.

Build Timing

Yes

Turns on build timing. If selected, the time it takes for the build to complete is posted to the Output window. For more information, see [Output Window](#).

No

Turns off build timing.

Maximum concurrent C++ compilations

Specifies the maximum number of CPU cores to use for parallel C++ compilation.

Extensions to Include

Specifies the file name extensions of files that can be ported into your project.

Extensions to Hide

Specifies the file name extensions of files that will not be displayed in **Solution Explorer** when **Show All Files** is enabled.

Build Customization Search Path

Specifies the list of directories that contain .rules files, which help you define build rules for your projects.

Solution Explorer Mode

Show only files in project

Configures **Solution Explorer** to only display files in the project.

Show all files

Configures **Solution Explorer** to show files in the project and files on disk in the project folder.

Enable Project Caching

Yes

Enables Visual Studio to cache project data so that when you open the project the next time, it can load that cached data rather than re-computing it from the project files. Using cached data can speed up the project load time significantly.

No

Do not use cached project data. Parse the project files each time the project loads.

See also

[Building C/C++ Programs](#)

[C/C++ Building Reference](#)

XAML Designer options page

1/30/2018 • 4 min to read • [Edit Online](#)

Use the **XAML Designer** options page to specify how elements and attributes are formatted in your XAML documents. To open this page, choose the **Tools** menu and then choose **Options**. To access the **XAML Designer** property page, choose the **XAML Designer** node. Settings for the XAML Designer are applied when you open the document. So, if you make changes to the settings, you need to close and then reopen Visual Studio to see the changes.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Personalize the Visual Studio IDE](#).

Enable XAML Designer

When selected, this setting enables the XAML Designer. The XAML Designer provides a visual work area for you to edit XAML documents. Certain functionality in Visual Studio, such as IntelliSense for resources and databinding, require the XAML Designer to be enabled.

The following settings apply only when XAML Designer is enabled. If you change this option, you will need to restart Visual Studio for the setting to take effect.

Default document view

Use this setting to control whether Design view appears when XAML documents are loaded.

| | |
|--------------------|---|
| Source View | Specifies whether only XAML source appears in the XAML view. This is useful when loading large documents. |
| Design View | Specifies whether only a visual XAML Designer appears in the XAML view. |
| Split View | Specifies whether both the visual XAML Designer and the XAML source appear next to one another in the XAML view (location based on the Split Orientation setting). |

Split Orientation

Use this setting to control when and how the XAML Designer appears when editing a XAML document. These settings apply only when **Default document view** is set to **Split View**.

| | |
|-----------------|---|
| Vertical | XAML source appears on the left side of the XAML view, and the XAML Designer appears on the other side. |
|-----------------|---|

| | |
|-------------------|--|
| Horizontal | The XAML Designer appears on the top of the XAML view, and the XAML source appears below it. |
| Default | The XAML document uses the split orientation recommended for the platform targeted by the document's project. For most platforms this is equivalent to Horizontal . |

Zoom by using

Use this setting to determine how zoom works when editing a XAML document.

| | |
|---------------------------|---|
| Mouse wheel | Zoom in the XAML Designer by scrolling the mouse wheel. |
| Ctrl + mouse wheel | Zoom in the XAML Designer by pressing the CTRL key while scrolling the mouse wheel. |
| Alt + mouse wheel | Zoom in the XAML Designer by pressing the ALT key while scrolling the mouse wheel. |

These settings determine Designer behavior when editing a XAML document.

| | |
|---|--|
| Automatically name interactive elements on creation | Specifies whether a default name is provided for a new interactive element when you add one to the Designer. |
| Automatically insert layout properties on element creation | Specifies whether layout properties are provided for a new element when you add one to the Designer. |
| Use quadrant based layout | Specifies whether the currently selected control aligns to the nearest edges of the parent container. If this checkbox is cleared, control alignments do not change during a move or create operation. |
| Automatically populate toolbox items | Specifies whether user controls and custom controls in the current solution are shown in the Toolbox automatically. |

Settings (Blend only)

Use these options to determine settings when editing XAML files using Blend.

| | |
|----------------------|--|
| Zoom by using | Zoom in the XAML Designer by scrolling the mouse wheel, or by pressing the CTRL or ALT key while scrolling the mouse wheel. |
| Type units | Specifies whether measurements on the designer are based on points or pixels. Because Universal Windows Apps don't support points, units are automatically converted to pixels if Points is selected. |

Artboard (Blend only)

Use these settings to determine XAML Designer behavior when editing XAML documents in Blend.

Snapping

| | |
|--------------------------|---|
| Show snap grid | When this option is selected, gridlines appear in the designer to help you align controls. Controls added to the designer snap to these gridlines when the Snap to gridlines option is selected. |
| Snap to gridlines | When controls are added or moved around the designer, they snap to the gridlines. |
| Gridline spacing | Specifies the spacing between the gridlines in either pixels or points (as determined by the Type units setting). |
| Snap to snaplines | Specifies whether controls snap to snaplines. |
| Default margin | When Snap to snaplines is enabled, specifies the spacing between the control and the snaplines in either pixels or points (as determined by the Type units setting). |
| Default padding | When Snap to snaplines is enabled, specifies the extra spacing between the control and the snaplines in either pixels or points (as determined by the Type units setting). |

Animation

Use this setting to determine whether a warning appears when dependent (non-accelerated) animations are enabled in Blend.

Effects

Use these settings to determine whether effects are rendered when editing XAML files in the XAML Designer using Blend.

| | |
|-----------------------|---|
| Render effects | Specifies whether effects render when editing XAML files in the XAML Designer using Blend. |
| Zoom threshold | Specifies the percentage of zoom in which effects render when the Render effects checkbox is selected. If you zoom beyond this setting, effects no longer render in the XAML Designer. |

See also

[XAML in WPF](#)

[Walkthrough: My first WPF desktop application](#)

Output Window

12/22/2017 • 3 min to read • [Edit Online](#)

The **Output** window can display status messages for various features in the integrated development environment (IDE). To open the **Output** window, on the menu bar, choose **View/Output** (or click CTRL + ALT + O).

WARNING

The Output window does not appear on the View menu in Visual Studio Express editions. To bring it up, use the hotkey CTRL + ALT + O.

Toolbar

Show output from

Displays one or more output panes to view. Several panes of information might be available, depending on which tools in the IDE have used the **Output** window to deliver messages to the user.

Find Message in Code

Moves the insertion point in the code editor to the line that contains the selected build error.

Go to Previous Message

Changes the focus in the **Output** window to the previous build error and moves the insertion point in the code editor to the line that contains that build error.

Go to Next Message

Changes the focus in the **Output** window to the next build error and moves the insertion point in the code editor to the line that contains that build error.

Clear all

Clears all text from the **Output** pane.

Toggle Word Wrap

Turns the Word Wrap feature on and off in the **Output** pane. When Word Wrap is on, text in longer entries that extends beyond the viewing area is displayed on the following line.

Output Pane

The **Output** pane selected in the **Show output from** list displays output from the source indicated.

Routing Messages to the Output Window

To display the **Output** window whenever you build a project, in the **General, Projects and Solutions, Options** dialog box, select **Show Output window when build starts**. Then, with a code file open for editing, choose the **Go to Next Message** and **Go To Previous Message** buttons on the **Output** window toolbar to select entries in the **Output** pane. As you do this, the insertion point in the code editor jumps to the line of code where the selected problem occurs.

Certain IDE features and commands invoked in the **Command Window** deliver their output to the **Output** window. Output from external tools such as .bat and .com files, which is typically displayed in the Command Prompt window, is routed to an **Output** pane when you select the **Use Output Window** option in the **Managing External Tools**. Many other kinds of messages can be displayed in **Output** panes as well. For example, when

Transact-SQL syntax in a stored procedure is checked against a target database, the results are displayed in the **Output** window.

You can also program your own applications to write diagnostic messages at run time to an **Output** pane. To do this, use members of the [Debug](#) class or [Trace](#) class in the [System.Diagnostics](#) namespace of the .NET Framework Class Library. Members of the [Debug](#) class display output when you build Debug configurations of your solution or project; members of the [Trace](#) class display output when you build either Debug or Release configurations. For more information, see [Diagnostic Messages in the Output Window](#).

In Visual C++, you can create custom build steps and build events whose warnings and errors are displayed and counted in the **Output** pane. By pressing F1 on a line of output, you can display an appropriate help topic. For more information, see [Formatting the Output of a Custom Build Step or Build Event](#).

Scrolling Behavior

If you use autoscrolling in the Output window and then navigate by using the mouse or arrow keys, autoscrolling stops. To resume autoscrolling, press CTRL+END.

See Also

[Diagnostic Messages in the Output Window](#)

[How to: Control the Output Window](#)

[Compiling and Building](#)

[Understanding Build Configurations](#)

[Class Library Overview](#)

Project Properties Reference

12/22/2017 • 1 min to read • [Edit Online](#)

Learn more about how to configure and customize project properties.

Project Properties Pages

| TITLE | DESCRIPTION |
|---|--|
| Application Page, Project Designer (Visual Basic) | Use this page to specify application settings and properties for a Visual Basic project. |
| Application Page, Project Designer (C#) | Use this page to specify application settings and properties for a Visual C# project. |
| Build Events Page, Project Designer (C#) | Use this pane to specify build configuration instructions. |
| Build Page, Project Designer (C#) | Use this pane to specify build configuration properties for a Visual C# project. |
| Compile Page, Project Designer (Visual Basic) | Use this page to specify compilation properties for Visual Basic projects. |
| Debug Page, Project Designer | Use this page to specify debugging properties for a project. |
| Code Analysis, Project Designer | Use this page to configure the code analysis tool. |
| Publish Page, Project Designer | Use this page to configure properties for ClickOnce. |
| References Page, Project Designer (Visual Basic) | Use this page to manage references used by a project. |
| Security Page, Project Designer | Use this page to configure code access security settings for applications that are deployed by using ClickOnce deployment. |
| Signing Page, Project Designer | Use this page to sign application and deployment manifests, and sign the assembly. |

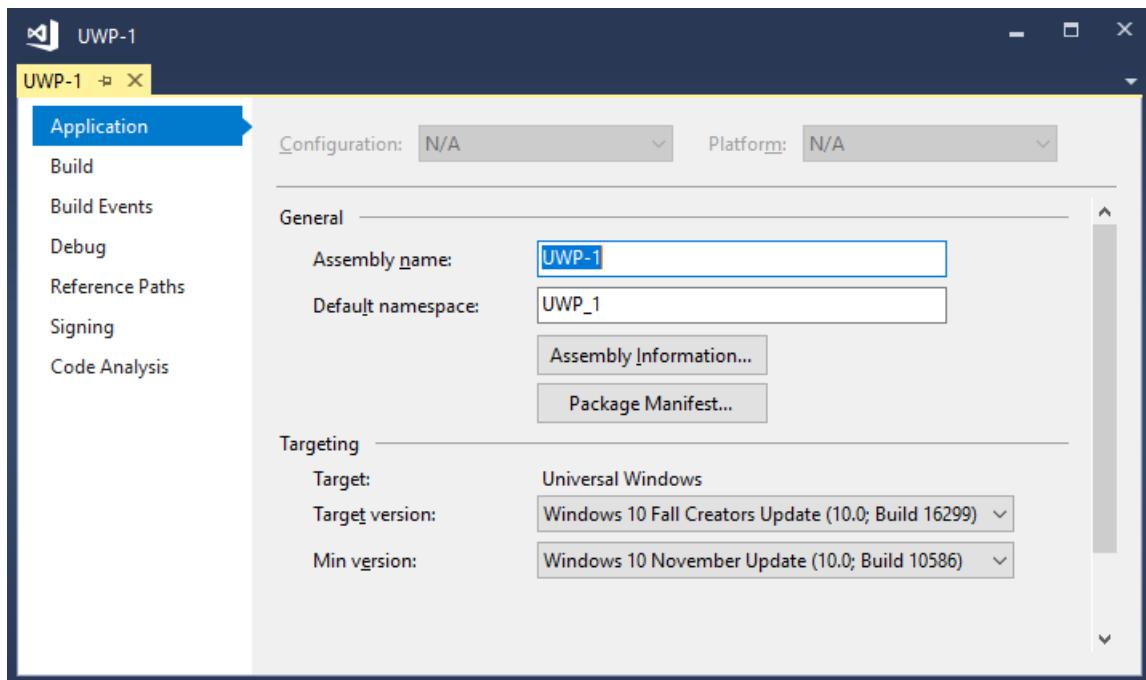
See Also

[Solutions and Projects](#)

Application property page (UWP projects)

1/25/2018 • 1 min to read • [Edit Online](#)

Use the **Application** property page to specify the Universal Windows Platform (UWP) project's assembly and package information, and target Windows 10 version.



To access the **Application** page, choose the project node in **Solution Explorer**. Then choose **Project > Properties** on the menu bar. The property pages open on the **Application** tab.

General section

Assembly name—Specifies the name of the output file that will hold the assembly manifest.

To access this property programmatically, see [AssemblyName](#).

Default namespace—Specifies the base namespace for files added to the project. For more information about namespaces, see [Namespaces \(C# programming guide\)](#), [Namespaces \(Visual Basic\)](#), or [Namespaces \(C++\)](#).

To access this property programmatically, see [RootNamespace](#).

Assembly Information—Choosing this button displays the [Assembly Information dialog box](#).

Package Manifest—Choosing this button opens the manifest designer. The manifest designer can also be accessed by choosing the *Package.appxmanifest* file in **Solution Explorer**. For more information, see [Configure a package with the manifest designer](#).

Targeting section

You can set the target version and minimum version of Windows 10 for your app by using the drop-down lists in this section. It is recommended that you target the latest version of Windows 10, and if you are developing an enterprise app, that you support an older minimum version too. For more information about which Windows 10 version to choose, see [Choose a UWP version](#).

For information about platform targeting in Visual Studio 2017, see [Platform targeting](#).

See also

[Create your first UWP app](#)

[Choose a UWP version](#)

Application Page, Project Designer (Visual Basic)

1/26/2018 • 9 min to read • [Edit Online](#)

Use the **Application** page of the Project Designer to specify a project's application settings and properties.

To access the **Application** page, choose a project node (not the **Solution** node) in **Solution Explorer**. Then choose **Project, Properties** on the menu bar. When the Project Designer appears, click the **Application** tab.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

General Application Settings

The following options enable you to configure general settings for an application.

Assembly name

Specifies the name of the output file that will contain the assembly manifest. If you change this property, the **Output Name** property will also change. You can also make this change at a command prompt by using [/out \(Visual Basic\)](#). For information about how to access this property programmatically, see [AssemblyName](#).

Root namespace

Specifies the base namespace for all files in the project. For example, if you set the **Root Namespace** to `Project1` and you have a `Class1` outside of any namespace in your code, its namespace would be `Project1.Class1`. If you have a `Class2` in a namespace `Order` in code, its namespace would be `Project1.Order.Class2`.

If you clear the **Root Namespace**, you can specify the namespace structure of your project in code.

NOTE

If you use the `Global` keyword in a [Namespace Statement](#), you can define a namespace out of the root namespace of your project. If you clear the **Root Namespace**, `Global` becomes the top-level namespace, which removes the need for the `Global` keyword in a [Namespace](#) statement. For more information, see "Global Keyword in Namespace Statements" in [Namespaces in Visual Basic](#).

For information about how to create namespaces in your code, see [Namespace Statement](#).

For more information about the root namespace property, see [/rootnamespace](#).

For information about how to access this property programmatically, see [RootNamespace](#).

Target framework (all configurations)

Specifies the version of the .NET Framework that the application targets. This option can have different values depending on which versions of the .NET Framework are installed on your computer.

The default value matches the target framework that you specified in the **New Project** dialog box.

NOTE

The prerequisite packages that are listed in the [Prerequisites Dialog Box](#) are set automatically when you open the dialog box for the first time. If you subsequently change the project's target framework, you must specify the prerequisites manually to match the new target framework.

For more information, see [How to: Target a Version of the .NET Framework](#) and [Visual Studio Multi-Targeting Overview](#).

Application type

Specifies the type of application to build. For Windows 8.x apps, you can specify **Windows Store App**, **Class Library**, or **WinMD File**. For most other application types, you can specify **Windows Application**, **Console Application**, **Class Library**, **Windows Service**, or **Web Control Library**.

For a web application project, you must specify **Class Library**.

If you specify the **WinMD File** option, types can be projected into any Windows Runtime programming language. By packaging the project's output as a WinMD file, you can code an application in multiple languages and have code interoperate as if you wrote it all in the same language. You can use the **WinMD File** option for solutions that target the Windows Runtime libraries, including Windows 8.x Store apps. For more information, see [Creating Windows Runtime Components in C# and Visual Basic](#).

NOTE

The Windows Runtime can project types so that they appear as native objects in whichever language uses them. For example, JavaScript applications that interact with Windows Runtime use it as a set of JavaScript objects, and C# applications use the library as a collection of .NET objects. By packaging the project's output as a WinMD file, you can take advantage of the same technology that Windows Runtime uses.

For more information about the **Application type** property, see [/target \(Visual Basic\)](#). For information about how to access that property programmatically, see [OutputType](#).

Icon

Sets the .ico file that you want to use as your program icon. Select <**Browse...**> to browse for an existing graphic. See [/win32icon](#) (or [/win32icon \(C# Compiler Options\)](#)) for more information. To access this property programmatically, see [ApplicationIcon](#).

Startup form / Startup object / Startup URI

Specifies the application's startup form or entry point.

If **Enable application framework** is selected (the default), this list is titled **Startup form** and shows only forms because the application framework supports only startup forms, not objects.

If the project is a WPF Browser Application, this list is titled **Startup URI**, and the default is **Page1.xaml**. The **Startup URI** list enables you to specify the user interface resource (a XAML element) that the application displays when the application starts. For more information, see [StartupUri](#).

If **Enable application framework** is cleared, this list becomes **Startup object** and shows both forms and classes or modules with a `Sub Main`.

Startup object defines the entry point to be called when the application loads. Generally this is set to either the main form in your application or to the `Sub Main` procedure that should run when the application starts. Because class libraries do not have an entry point, their only option for this property is **(None)**. For more information, see [/main](#). To access this property programmatically, see [StartupObject](#).

Assembly Information

Click this button to display the [Assembly Information Dialog Box](#).

Enable application framework

Specifies whether a project will use the application framework. The setting of this option affects the options available in **Startup form/Startup object**.

If this check box is selected, your application uses the standard `Sub Main`. Selecting this check box enables the features in the **Windows application framework properties** section, and also requires you to select a startup form.

If this check box is cleared, your application uses the custom `Sub Main` that you specified in **Startup form**. In this case you can specify either a startup object (a custom `Sub Main` in a method or a class) or a form. Also, the options in the **Windows application framework properties** section become unavailable.

View Windows Settings

Click this button to generate and open the app.manifest file. Visual Studio uses this file to generate manifest data for the application. Then set the UAC requested execution level by modifying the `<requestedExecutionLevel>` tag in app.manifest as follows:

```
<requestedExecutionLevel level="asInvoker" />
```

ClickOnce works with a level of `asInvoker` or in virtualized mode (no manifest generation). To specify virtualized mode, remove the entire tag from app.manifest.

For more information about manifest generation, see [ClickOnce Deployment on Windows Vista](#).

Windows Application Framework Properties

The following settings are available in the **Windows application framework properties** section. These options are available only if the **Enable application framework** check box is selected. The section following this one describes **Windows application framework properties** settings for Windows Presentation Foundation (WPF) Applications.

Enable XP visual styles

Enables or disables the Windows XP visual styles, also known as *Windows XP Themes*. Windows XP visual styles enable, for example, controls with rounded corners and dynamic colors. The default is enabled.

Make single instance application

Select this check box to prevent users from running multiple instances of the application. The default setting for this check box is cleared. This setting allows multiple instances of the application to be run.

Save My.Settings on Shutdown

Select this check box to specify that the application's `My.Settings` settings are saved when users shut down their computers. The default setting is enabled. If this option is disabled, you can save application settings manually by calling `My.Settings.Save`.

Authentication mode

Select **Windows** (the default) to specify the use of Windows authentication to identify the currently logged-on user. You can retrieve this information at run time by using the `My.User` object. Select **Application-defined** if you will provide your own code to authenticate users instead of using the default Windows authentication methods.

Shutdown mode

Select **When startup form closes** (the default) to specify that the application exit when the form set as the startup form closes, even if other forms are open. Select **When last form closes** to specify that the application exit when the last form is closed or when `My.Application.Exit` or the `End` statement is called explicitly.

Select **On explicit shutdown** to specify that the application exit when you explicitly call `Shutdown`.

Select **On last window close** to specify that the application exit when the last window closes or when you explicitly call `Shutdown`. This is the default setting.

Select **On main window close** to specify that the application exit when the main window closes or when you explicitly call `Shutdown`.

Splash screen

Select the form that you want to use as a splash screen. You must have previously created a splash screen by using a form or a template. The default is **(None)**.

View Application Events

Click this button to display an events code file in which you can write events for the application framework events `Startup`, `Shutdown`, `UnhandledException`, `StartupNextInstance` and `NetworkAvailabilityChanged`. You can also override certain application framework methods. For example, you can change the display behavior of the splash screen by overriding `OnInitialize`.

Windows Application Framework Properties for Windows Presentation Foundation (WPF) Applications

The following settings are available in the **Windows application framework properties** section when the project is a Windows Presentation Foundation application. These options are available only if the **Enable application framework** check box is selected. The options listed in this table are available only for WPF applications or WPF browser applications. They are not available for WPF User Control or Custom Control libraries.

Shutdown mode

This property is applicable only to Windows Presentation Foundation applications.

Select **On explicit shutdown** to specify that the application exit when you explicitly call `Shutdown`.

Select **On last window close** to specify that the application exit when the last window closes or when you explicitly call `Shutdown`. This is the default setting.

Select **On main window close** to specify that the application exit when the main window closes or when you explicitly call `Shutdown`.

For more information about using this setting, see [Shutdown](#)

Edit XAML

Click this button to open and modify the application definition file (`Application.xaml`) in the XAML editor. When you click this button, `Application.xaml` opens at the application definition node. You might have to edit this file to perform certain tasks, such as defining resources. If the application definition file does not exist, the Project Designer creates one.

View Application Events

Click this button to display the `Application` partial class file (`Application.xaml.vb`) in a code editor. If the file does not exist, the Project Designer creates one with the appropriate class name and namespace.

The `Application` object raises events when certain application state changes occur (for example, on application startup or shutdown). For a full list of the events that this class exposes, see [Application](#). These events are handled in the user code section of the `Application` partial class.

Assembly Information Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

The **Assembly Information** dialog box is used to specify the values of the .NET Framework global assembly attributes, which are stored in the AssemblyInfo file created automatically with your project. In **Solution Explorer**, the file is located in the **My Project** node in Visual Basic (click **Show All files** to view it); it is located under **Properties** in Visual C#. For more information about assembly attributes, see [Attributes](#).

To access this dialog box, select a project node in **Solution Explorer**, and then, on the **Project** menu, click **Properties**. When the **Project Designer** appears, click the **Application** tab. On the **Application** page, click the **Assembly Information** button.

UIElement List

Title

Specifies a title for the assembly manifest. Corresponds to [AssemblyTitleAttribute](#).

Description

Specifies an optional description for the assembly manifest. Corresponds to [AssemblyDescriptionAttribute](#).

Company

Specifies a company name for the assembly manifest. Corresponds to [AssemblyCompanyAttribute](#).

Product

Specifies a product name for the assembly manifest. Corresponds to [AssemblyProductAttribute](#).

Copyright

Specifies a copyright notice for the assembly manifest. Corresponds to [AssemblyCopyrightAttribute](#).

Trademark

Specifies a trademark for the assembly manifest. Corresponds to [AssemblyTrademarkAttribute](#).

Assembly Version

Specifies the version of the assembly. Corresponds to [AssemblyVersionAttribute](#).

File Version

Specifies a version number that instructs the compiler to use a specific version for the Win32 file version resource. Corresponds to [AssemblyFileVersionAttribute](#).

GUID

A unique GUID that identifies the assembly. When you create a project, Visual Studio generates a GUID for the assembly. Corresponds to [Guid](#).

Neutral Language

Specifies which culture the assembly supports. Corresponds to [NeutralResourcesLanguageAttribute](#). The default is **(None)**.

Make assembly COM-Visible

Specifies whether types in the assembly will be available to COM. Corresponds to [ComVisibleAttribute](#).

See Also

[Application Page, Project Designer \(Visual Basic\)](#)

[Attributes](#)

Application Page, Project Designer (C#)

1/26/2018 • 4 min to read • [Edit Online](#)

Use the **Application** page of the **Project Designer** to specify the project's application settings and properties.

To access the **Application** page, choose a project node (not the **Solution** node) in **Solution Explorer**. Then choose **Project, Properties** on the menu bar. When the Project Designer appears, click the **Application** tab.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

General Application Settings

The following options enable you to configure general settings for the application.

Assembly name

Specifies the name of the output file that will hold the assembly manifest. Changing this property will also change the **Output Name** property. You can also make this change from the command line by using [/out \(C# Compiler Options\)](#). To access this property programmatically, see [AssemblyName](#).

Default namespace

Specifies the base namespace for files added to the project.

See [namespace](#) for more information about creating namespaces in your code.

To access this property programmatically, see [RootNamespace](#).

Target Framework

Specifies the version of the .NET Framework that the application targets. This option can have different values depending on which versions of the .NET Framework are installed on your computer.

By default, the value is the same as the target framework that you selected in the **New Project** dialog box.

NOTE

The prerequisite packages listed in the [Prerequisites Dialog Box](#) are set automatically the first time that you open the dialog box. If you subsequently change the project's target framework, you will have to select the prerequisites manually to match the new target framework.

For more information, see [How to: Target a Version of the .NET Framework](#) and [Visual Studio Multi-Targeting Overview](#).

Application type

Specifies the type of application to build. For Windows 8.x apps, you can specify **Windows Store App**, **Class Library**, or **WinMD File**. For most other application types, you can specify **Windows Application**, **Console Application**, **Class Library**, **Windows Service**, or **Web Control Library**.

For a web application project, you must specify **Class Library**.

If you specify the **WinMD File** option, types can be projected into any Windows Runtime programming language.

By packaging the project's output as a WinMD file, you can code an application in multiple languages and have code interoperate as if you wrote it all in the same language. You can specify this option for solutions that target the Windows Runtime libraries, including Windows 8.x Store apps. For more information, see [Creating Windows Runtime Components in C# and Visual Basic](#).

NOTE

The Windows Runtime can project types so that they appear as native objects in whichever language uses them. For example, JavaScript applications that interact with Windows Runtime use it as a set of JavaScript objects, and C# applications use the library as a collection of .NET objects. By packaging the project's output as a WinMD file, you can take advantage of the same technology that Windows Runtime uses.

For more information about the **Application type** property, see [/target \(C# Compiler Options\)](#). For information about how to access this property programmatically, see [OutputType](#).

Assembly Information

Clicking this button displays the [Assembly Information Dialog Box](#).

Startup object

Defines the entry point to be called when the application loads. Generally this is set either to the main form in your application or to the `Main` procedure that should run when the application starts. Because class libraries do not have an entry point, their only option for this property is **(Not Set)**.

By default, in a WPF Browser Application project, this option is **(Not set)**. The other option is `Projectname.App`. In this kind of project, you have to set the startup URI to load a UI resource when the application starts. To do this, open the Application.xaml file in your project and set the `StartupUri` property to a .xaml file in your project, such as Window1.xaml. For a list of acceptable root elements, see [StartupUri](#). You also have to define a `public static void Main()` method in a class in the project. This class will appear in the **Startup object** list as `ProjectName.ClassName`. You can then select the class as the startup object.

See [/main \(C# Compiler Options\)](#) for more information. To access this property programmatically, see [StartupObject](#).

Resources

The following options enable you to configure general settings for the application.

Icon and manifest

By default, this radio button is selected and the **Icon** and **Manifest** options are enabled. This enables you to select your own icon, or to select different manifest generation options. Leave this radio button selected unless you are providing a resource file for the project.

Icon

Sets the .ico file that you want to use as your program icon. Click the ellipsis button to browse for an existing graphic, or type the name of the file that you want. See [/win32icon \(C# Compiler Options\)](#) for more information. To access this property programmatically, see [ApplicationIcon](#).

Manifest

Selects a manifest generation option when the application runs on Windows Vista under User Account Control (UAC). This option can have the following values:

- **Embed manifest with default settings.** Supports the typical manner in which Visual Studio operates on Windows Vista, which is to embed security information in the application's executable file, specifying that `requestedExecutionLevel` be `AsInvoker`. This is the default option.
- **Create application without a manifest.** This method is known as *virtualization*. Use this option for

compatibility with earlier applications.

- **Properties\app.manifest**. This option is required for applications deployed by ClickOnce or Registration-Free COM. If you publish an application by using ClickOnce deployment, **Manifest** is automatically set to this option.

Resource File

Select this radio button when you are providing a resource file for the project. Selecting this option disables the **Icon** and **Manifest** options.

Enter a path name or use the Browse button (...) to add a Win32 resource file to the project.

Build Events Page, Project Designer (C#)

12/22/2017 • 1 min to read • [Edit Online](#)

Use the **Build Events** page of the **Project Designer** to specify build configuration instructions. You can also specify the conditions under which any post-build events are run. For more information, see [How to: Specify Build Events \(C#\)](#) and [How to: Specify Build Events \(Visual Basic\)](#).

UIElement List

Configuration

This control is not editable in this page. For a description of this control, see [Build Page, Project Designer \(C#\)](#).

Platform

This control is not editable on this page. For a description of this control, see [Build Page, Project Designer \(C#\)](#).

Pre-build event command line

Specifies any commands to execute before the build starts. To type long commands, click **Edit Pre-build** to display the [Pre-build Event/Post-build Event Command Line Dialog Box](#).

NOTE

Pre-build events do not run if the project is up to date and no build is triggered.

Post-build event command line

Specifies any commands to execute after the build ends. To type long commands, click **Edit Post-build** to display the [Pre-build Event/Post-build Event Command Line Dialog Box](#).

NOTE

Add a `call` statement before all post-build commands that run .bat files. For example, `call C:\MyFile.bat` or
`call C:\MyFile.bat call C:\MyFile2.bat`.

Run the post-build event

Specifies the following conditions for the post-build event to run, as shown in the following table.

| OPTION | RESULT |
|--|--|
| Always | Post-build event will run regardless of whether the build succeeds. |
| On successful build | Post-build event will run if the build succeeds. Thus, the event will run even for a project that is up-to-date, as long as the build succeeds. |
| When the build updates the project output | Post-build event will only run when the compiler's output file (.exe or .dll) is different than the previous compiler output file. Thus, a post-build event is not run if a project is up-to-date. |

See Also

[How to: Specify Build Events \(Visual Basic\)](#)

[How to: Specify Build Events \(C#\)](#)

[Project Properties Reference](#)

[Compiling and Building](#)

Pre-build Event/Post-build Event Command Line Dialog Box

12/22/2017 • 2 min to read • [Edit Online](#)

You can type pre- or post-build events for the [Build Events Page](#), [Project Designer \(C#\)](#) directly in the edit box, or you can select pre- and post-build macros from a list of available macros.

NOTE

Pre-build events do not run if the project is up to date and no build is triggered.

UI Element List

Command line edit box

Contains the events to run either for pre-build or post-build.

NOTE

Add a `call` statement before all post-build commands that run .bat files. For example, `call C:\MyFile.bat` or `call C:\MyFile.bat call C:\MyFile2.bat`.

Macros

Expands the edit box to display a list of macros to insert in the command line edit box.

Macro table

Lists the available macros and its value. See Macros below for a description of each. You can select only one macro at a time to insert into the command line edit box.

Insert

Inserts into the command line edit box the macro selected in the macro table.

Macros

You can use any of these macros to specify locations for files, or to get the actual name of the input file in the case of multiple selections. These macros are not case-sensitive.

| MACRO | DESCRIPTION |
|------------------------------------|---|
| <code>\$(ConfigurationName)</code> | The name of the current project configuration, for example, "Debug". |
| <code>\$(OutDir)</code> | Path to the output file directory, relative to the project directory. This resolves to the value for the Output Directory property. It includes the trailing backslash '\'. |
| <code>\$(DevEnvDir)</code> | The installation directory of Visual Studio (defined with drive and path); includes the trailing backslash '\'. |
| <code>\$(PlatformName)</code> | The name of the currently targeted platform. For example, "AnyCPU". |

| MACRO | DESCRIPTION |
|-----------------------------------|---|
| <code>\$(ProjectDir)</code> | The directory of the project (defined with drive and path); includes the trailing backslash '\'. |
| <code>\$(ProjectPath)</code> | The absolute path name of the project (defined with drive, path, base name, and file extension). |
| <code>\$(ProjectName)</code> | The base name of the project. |
| <code>\$(ProjectFileName)</code> | The file name of the project (defined with base name and file extension). |
| <code>\$(ProjectExt)</code> | The file extension of the project. It includes the '.' before the file extension. |
| <code>\$(SolutionDir)</code> | The directory of the solution (defined with drive and path); includes the trailing backslash '\'. |
| <code>\$(SolutionPath)</code> | The absolute path name of the solution (defined with drive, path, base name, and file extension). |
| <code>\$(SolutionName)</code> | The base name of the solution. |
| <code>\$(SolutionFileName)</code> | The file name of the solution (defined with base name and file extension). |
| <code>\$(SolutionExt)</code> | The file extension of the solution. It includes the '.' before the file extension. |
| <code>\$(TargetDir)</code> | The directory of the primary output file for the build (defined with drive and path). It includes the trailing backslash '\'. |
| <code>\$(TargetPath)</code> | The absolute path name of the primary output file for the build (defined with drive, path, base name, and file extension). |
| <code>\$(TargetName)</code> | The base name of the primary output file for the build. |
| <code>\$(TargetFileName)</code> | The file name of the primary output file for the build (defined as base name and file extension). |
| <code>\$(TargetExt)</code> | The file extension of the primary output file for the build. It includes the '.' before the file extension. |

See Also

- [Specifying Custom Build Events in Visual Studio](#)
- [Build Events Page, Project Designer \(C#\)](#)
- [How to: Specify Build Events \(Visual Basic\)](#)
- [How to: Specify Build Events \(C#\)](#)

Build Page, Project Designer (C#)

1/26/2018 • 5 min to read • [Edit Online](#)

Use the **Build** page of the **Project Designer** to specify the project's build configuration properties. This page applies to Visual C# projects only.

To access the **Build** page, choose a project node (not the **Solution** node) in **Solution Explorer**. Then choose **View, Property Pages** on the menu. When the Project Designer appears, choose the **Build** tab.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

Configuration and Platform

The following options enable you to select the configuration and platform to display or modify.

NOTE

With simplified build configurations, the project system determines whether to build a debug or release version. Therefore, these options are not displayed. For more information, see [How to: Set debug and release configurations](#).

Configuration

Specifies which configuration settings to display or modify. The settings can be **Active (Debug)** (this is the default), **Debug**, **Release**, or **All Configurations**.

Platform

Specifies which platform settings to display or modify. The default setting is **Active (Any CPU)**. You can change the active platform using the **Configuration Manager**. For more information, see [How to: Create and Edit Configurations](#).

General

The following options enable you to configure several C# compiler settings.

Conditional compilation symbols

Specifies symbols on which to perform conditional compilation. Separate symbols with a semi-colon (";"). For more information, see [/define \(C# Compiler Options\)](#).

Define DEBUG constant

Defines DEBUG as a symbol in all source code files in your app. Selecting this is equivalent to using the `/define:DEBUG` command line option.

Define TRACE constant

Defines TRACE as a symbol in all source code files in your app. Selecting this is equivalent to using the `/define:TRACE` command line option.

Platform Target

Specifies the processor to be targeted by the output file. Choose **x86** for any 32-bit Intel-compatible processor,

choose **x64** for any 64-bit Intel-compatible processor, choose **ARM** for ARM processors, or choose **Any CPU** to specify that any processor is acceptable. **Any CPU** is the default value for projects, because it allows the application to run on the broadest range of hardware.

For more information, see [/platform \(C# Compiler Options\)](#).

Prefer 32-bit

If the **Prefer32-bit** check box is selected, the application runs as a 32-bit application on both 32-bit and 64-bit versions of Windows. If the check box is cleared, the application runs as a 32-bit application on 32-bit versions of Windows and as a 64-bit application on 64-bit versions of Windows.

If you run an application as a 64-bit application, the pointer size doubles, and compatibility problems can occur with other libraries that are exclusively 32-bit. It is useful to run a 64-bit application only if it needs more than 4 GB of memory or 64-bit instructions provide a significant performance improvement.

This check box is available only if all of the following conditions are true:

- On the **Build Page**, the **Platform target** list is set to **Any CPU**.
- On the **Application Page**, the **Output type** list specifies that the project is an application.
- On the **Application Page**, the **Target framework** list specifies the .NET Framework 4.5.

Allow unsafe code

Allows code that uses the **unsafe** keyword to compile. For more information, see [/unsafe \(C# Compiler Options\)](#).

Optimize code

Enable or disable optimizations performed by the compiler to make your output file smaller, faster, and more efficient. For more information, see [/optimize \(C# Compiler Options\)](#).

Errors and Warnings

The following settings are used to configure the error and warning options for the build process.

Warning level

Specifies the level to display for compiler warnings. For more information, see [/warn \(C# Compiler Options\)](#).

Suppress warnings

Blocks the compiler's ability to generate one or more warnings. Separate multiple warning numbers with a comma or semicolon. For more information, see [/nowarn \(C# Compiler Options\)](#).

Treat Warnings as Errors

The following settings are used to specify which warnings are treated as errors. Select one of the following options to indicate under what conditions to return an error when the build encounters a warning. For more information, see [/warnaserror \(C# Compiler Options\)](#).

None

Treats no warnings as errors.

Specific warnings

Treats the specified warnings as errors. Separate multiple warning numbers with a comma or semicolon.

All

Treats all warnings as errors.

Output

The following settings are used to configure the output options for the build process.

Output path

Specifies the location of the output files for this project's configuration. Enter the path of the build output in this box, or choose the **Browse** button to specify a path. Note that the path is relative; if you enter an absolute path, it will be saved as relative. The default path is bin\Debug or bin\Release\.

With simplified build configurations, the project system determines whether to build a debug or release version. The **Build** command from the **Debug** menu (F5) will put the build in the debug location regardless of the **Output path** you specify. However, the **Build** command from the **Build** menu puts it in the location you specify. For more information, see [Understanding Build Configurations](#).

XML documentation file

Specifies the name of a file into which documentation comments will be processed. For more information, see [/doc \(C# Compiler Options\)](#).

Register for COM interop

Indicates that your managed application will expose a COM object (a COM callable wrapper) that allows a COM object to interact with your managed application. The **Output type** property in the [Application page](#) of the **Project Designer** for this application must be set to **Class Library** in order for the **Register for COM interop** property to be available. For an example class that you might include in your Visual C# application and expose as a COM object, see [Example COM Class](#).

Generate serialization assembly

Specifies whether the compiler will use the XML Serializer Generator Tool (Sgen.exe) to create XML serialization assemblies. Serialization assemblies can improve the startup performance of [XmlSerializer](#) if you have used that class to serialize types in your code. By default, this option is set to **Auto**, which specifies that serialization assemblies be generated only if you have used [XmlSerializer](#) to encode types in your code to XML. **Off** specifies that serialization assemblies never be generated, regardless of whether your code uses [XmlSerializer](#). **On** specifies that serialization assemblies always be generated. Serialization assemblies are named `TypeName.XmlSerializers.dll`. For more information, see [XML Serializer Generator Tool \(Sgen.exe\)](#).

Advanced

Click to display the [Advanced Build Settings Dialog Box \(C#\)](#) dialog box.

See Also

[Project Properties Reference](#)

[C# Compiler Options](#)

Advanced Build Settings Dialog Box (C#)

12/22/2017 • 2 min to read • [Edit Online](#)

Use the **Advanced Build Settings** dialog box of the **Project Designer** to specify the project's advanced build configuration properties. This dialog box applies to Visual C# projects only.

General

The following options enable you to set general advanced settings.

Language Version Specifies the version of the language to use. The feature set is different in each version, so this option can be used to force the compiler to allow only a subset of the implemented features, or to enable only those features compatible with an existing standard. This setting has the following options:

- **default**

Targets the current version.

- **ISO-1 and ISO-2**

Targets the ISO-1 and ISO-2 standard features, respectively.

- **C# [version number]**

Targets a specific version of C#. For more information, see [/langversion \(C# Compiler Options\)](#).

Internal Compiler Error Reporting Specifies whether to report compiler errors to Microsoft. If set to **prompt** (the default), you will receive a prompt if an internal compiler error occurs, giving you the option of sending an error report electronically to Microsoft. If set to **send**, an error report will be sent automatically. If set to **queue**, error reports will be queued. If set to **none**, the error will be reported only in the compiler's text output. For more information, see [/errorreport \(C# Compiler Options\)](#).

Check for arithmetic overflow/underflow Specifies whether an integer arithmetic statement that is not in the scope of the **checked** or **unchecked** keywords and that results in a value outside the range of the data type will cause a run-time exception. For more information, see [/checked \(C# Compiler Options\)](#).

Do not reference mscorlib.dll Specifies whether mscorlib.dll will be imported into your program, defining the entire **System** namespace. Check this box if you want to define or create your own **System** namespace and objects. For more information, see [/nostdlib \(C# Compiler Options\)](#).

Output

The following options enable you to specify advanced output options.

Debug Information Specifies the type of debugging information generated by the compiler. For information on how to configure the debug performance of an application, see [Making an Image Easier to Debug](#). This setting has the following options:

- **none**

Specifies that no debugging information will be generated.

- **full**

Enables attaching a debugger to the running program.

- **pdbsonly**

Allows source code debugging when the program is started in the debugger but will only display assembler when the running program is attached to the debugger.

- **portable**

Produces a .PDB file, a non-platform-specific, portable symbol file that provides other tools, especially debuggers, information about what is in the main executable file and how it was produced. See [Portable PDB](#) for more information.

- **embedded**

Embeds portable symbol information into the assembly. No external .PDB file is produced.

For more information, see [/debug \(C# Compiler Options\)](#).

File Alignment Specifies the size of sections in the output file. Valid values are **512**, **1024**, **2048**, **4096**, and **8192**. These values are measured in bytes. Each section will be aligned on a boundary that is a multiple of this value, affecting the size of the output file. For more information, see [/filealign \(C# Compiler Options\)](#).

Library Base Address Specifies the preferred base address at which to load a DLL. The default base address for a DLL is set by the .NET Framework common language runtime. For more information, see [/baseaddress \(C# Compiler Options\)](#).

See Also

[C# Compiler Options Build Page](#), [Project Designer \(C#\)](#)

Code Analysis, Project Designer

12/22/2017 • 1 min to read • [Edit Online](#)

Contains the code analysis tool that you can opt to run on your code. The tool reports information about your assemblies, such as violations of the programming and design rules set forth in the Microsoft .NET Framework Design Guidelines.

UIElement List

Enable Code Analysis

Enables or disables code analysis for your project.

Design Rules

Enables or disables the design rules. You can also expand this entry to enable or disable individual rules.

Globalization Rules

Enables or disables the globalization rules. You can also expand this entry to enable or disable individual rules.

Interoperability Rules

Enables or disables the interoperability rules. You can also expand this entry to enable or disable individual rules.

Maintainability Rules

Enables or disables the maintainability rules. You can also expand this entry to enable or disable individual rules.

Mobility Rules

Enables or disables the mobility rules. You can also expand this entry to enable or disable individual rules.

Naming Rules

Enables or disables the naming rules. You can also expand this entry to enable or disable individual rules.

Performance Rules

Enables or disables the performance rules. You can also expand this entry to enable or disable individual rules.

Portability Rules

Enables or disables the portability rules. You can also expand this entry to enable or disable individual rules.

Reliability Rules

Enables or disables the reliability rules. You can also expand this entry to enable or disable individual rules.

Security Rules

Enables or disables the security rules. You can also expand this entry to enable or disable individual rules.

Usage Rules

Enables or disables the usage rules. You can also expand this entry to enable or disable individual rules.

See Also

[Code Analysis for Managed Code Warnings](#)

[Code Analysis for Managed Code Overview](#)

[Walkthrough: Analyzing Managed Code for Code Defects](#)

Compile Page, Project Designer (Visual Basic)

1/26/2018 • 8 min to read • [Edit Online](#)

Use the **Compile** page of the Project Designer to specify compilation instructions. You can also specify advanced compiler options and pre-build or post-build events on this page.

To access the **Compile** page, choose a project node (not the **Solution** node) in **Solution Explorer**. Then choose **Project, Properties** on the menu bar. When the Project Designer appears, click the **Compile** tab.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

Configuration and Platform

The following settings enable you to select the configuration and platform to display or modify.

NOTE

With simplified build configurations, the project system determines whether to build a debug or release version. Therefore, the **Configuration** and **Platform** lists are not displayed.

Configuration

Specifies which configuration settings to display or modify. The settings are **Debug** (default), **Release**, or **All Configurations**. For more information, see [Understanding Build Configurations](#) and [How to: Create and Edit Configurations](#).

Platform

Specifies which platform settings to display or modify. You can specify **Any CPU** (default), **x64**, or **x86**.

Compiler Configuration Options

The following settings enable you to set the compiler configuration options.

Build output path

Specifies the location of the output files for this project's configuration. Type the path of the build output in this box, or click the **Browse** button to select a path. Note that the path is relative; if you enter an absolute path, it will be saved as relative. The default path is bin\Debug\ or bin\Release\.

With simplified build configurations, the project system determines whether to build a debug or release version. The **Build** command from the **Debug** menu (F5) will put the build in the debug location regardless of the **Output path** you specify. However, the **Build** command from the **Build** menu puts it in the location you specify.

Option explicit

Specifies whether to allow implicit declaration of variables. Select **On** to require explicit declaration of variables. This causes the compiler to report errors if variables are not declared before they are used. Select **Off** to allow implicit declaration of variables.

This setting corresponds to the [/optionexplicit](#) compiler option.

If a source code file contains an [Option Explicit Statement](#), the [On](#) or [Off](#) value in the statement overrides the **Option Explicit** setting on the **Compile page**.

When you create a new project, the **Option Explicit** setting on the **Compile page** is set to the value of the **Option Explicit** setting in the **Options** dialog box. To view or change the setting in this dialog box, on the **Tools** menu, click **Options**. In the **Options** dialog box, expand **Projects and Solutions**, and then click **VB Defaults**. The initial default setting of **Option Explicit** in **VB Defaults** is **On**.

Setting **Option Explicit** to [Off](#) is generally not a good practice. You could misspell a variable name in one or more locations, which would cause unexpected results when the program is run.

Option strict

Specifies whether to enforce strict type semantics. When **Option Strict** is **On**, the following conditions cause a compile-time error:

- Implicit narrowing conversions
- Late binding
- Implicit typing that results in an [Object](#) type

Implicit narrowing conversion errors occur when there is an implicit data type conversion that is a narrowing conversion. For more information, see [Option Strict Statement](#), [Implicit and Explicit Conversions](#), and [Widening and Narrowing Conversions](#).

An object is late bound when it is assigned to a property or method of a variable that is declared to be of type [Object](#). For more information, see [Option Strict Statement](#) and [Early and Late Binding](#).

Implicit object type errors occur when an appropriate type cannot be inferred for a declared variable, so a type of [Object](#) is inferred. This primarily occurs when you use a [Dim](#) statement to declare a variable without using an [As](#) clause, and [Option Infer](#) is off. For more information, see [Option Strict Statement](#), [Option Infer Statement](#), and the [Visual Basic Language Specification](#).

The **Option Strict** setting corresponds to the [/optionstrict](#) compiler option.

If a source code file contains an [Option Strict Statement](#), the [On](#) or [Off](#) value in the statement overrides the **Option Strict** setting on the **Compile page**.

When you create a project, the **Option Strict** setting on the **Compile page** is set to the value of the **Option Strict** setting in the **Options** dialog box. To view or change the setting in this dialog box, on the **Tools** menu, click **Options**. In the **Options** dialog box, expand **Projects and Solutions**, and then click **VB Defaults**. The initial default setting of **Option Strict** in **VB Defaults** is **Off**.

Option Strict Individual Warnings. The **Warning configurations** section of the **Compile page** has settings that correspond to the three conditions that cause a compile-time error when [Option Strict](#) is on. Following are these settings:

- **Implicit conversion**
- **Late binding; call could fail at run time**
- **Implicit type; object assumed**

When you set **Option Strict** to **On**, all three of these warning configuration settings are set to **Error**. When you set **Option Strict** to **Off**, all three settings are set to **None**.

You can individually change each warning configuration setting to **None**, **Warning**, or **Error**. If all three warning configuration settings are set to **Error**, [On](#) appears in the [Option strict](#) box. If all three are set to **None**, [Off](#) appears in this box. For any other combination of these settings, **(custom)** appears.

Option compare

Specifies the type of string comparison to use. Select **Binary** to instruct the compiler to use binary, case-sensitive string comparisons. Select **Text** to use locale-specific, case-insensitive text string comparisons.

This setting corresponds to the [/optioncompare](#) compiler option.

If a source code file contains an [Option Compare Statement](#), the **Binary** or **Text** value in the statement overrides the **Option Compare** setting on the **Compile page**.

When you create a project, the **Option Compare** setting on the **Compile page** is set to the value of the **Option Compare** setting in the **Options** dialog box. To view or change the setting in this dialog box, on the **Tools** menu, click **Options**. In the **Options** dialog box, expand **Projects and Solutions**, and then click **VB Defaults**. The initial default setting of **Option Compare** in **VB Defaults** is **Binary**.

Option infer

Specifies whether to allow local type inference in variable declarations. Select **On** to allow the use of local type inference. Select **Off** to block local type inference.

This setting corresponds to the [/optioninfer](#) compiler option.

If a source code file contains an [Option Infer Statement](#), the **on** or **off** value in the statement overrides the **Option Infer** setting on the **Compile page**.

When you create a project, the **Option Infer** setting on the **Compile page** is set to the value of the **Option Infer** setting in the **Options** dialog box. To view or change the setting in this dialog box, on the **Tools** menu, click **Options**. In the **Options** dialog box, expand **Projects and Solutions**, and then click **VB Defaults**. The initial default setting of **Option Infer** in **VB Defaults** is **On**.

Target CPU

Specifies the processor to be targeted by the output file. Specify **x86** for any 32-bit Intel-compatible processor, **x64** for any 64-bit Intel-compatible processor, **ARM** for any ARM processor, or **Any CPU** to specify that any processor is acceptable. **Any CPU** is the default value for new projects because it allows the application to run on the largest number of hardware types.

For more information, see [/platform \(Visual Basic\)](#).

Prefer 32-bit

If the **Prefer32-bit** check box is selected, the application runs as a 32-bit application on both 32-bit and 64-bit versions of Windows. Otherwise, the application runs as a 32-bit application on 32-bit versions of Windows and as a 64-bit application on 64-bit versions of Windows.

Running as a 64-bit application doubles the pointer size, and it can cause compatibility problems with libraries that are exclusively 32-bit. It makes sense to run an application as 64-bit only if it runs significantly faster or needs more than 4 GB of memory.

This check box is available only if all of the following conditions are true:

- On the **Compile Page**, the **Target CPU** list is set to **Any CPU**.
- On the **Application Page**, the **Application type** list specifies that the project is an application.
- On the **Application Page**, the **Target framework** list specifies the .NET Framework 4.5.

Warning configurations

This table lists build conditions and the corresponding notification level of **None**, **Warning**, or **Error** for each.

By default, all compiler warnings are added to the Task List during compilation. Select **Disable all warnings** to instruct the compiler not to issue warnings or errors. Select **Treat all warnings as errors** if you want the compiler to treat warnings as errors that must be fixed.

Disable all warnings

Specifies whether to allow the compiler to issue notifications as specified in the **Condition and Notification** table described earlier in this document. By default, this check box is cleared. Select this check box to instruct the compiler not to issue warnings or errors.

This setting corresponds to the [/nowarn](#) compiler option.

Treat all warnings as errors

Specifies how to treat warnings. By default, this check box is cleared, so that all warning notifications remain set to **Warning**. Select this check box to change all warning notifications to **Error**.

This option is available only if **Disable all warnings** is cleared.

Generate XML documentation file

Specifies whether to generate documentation information. By default, this check box is selected, instructing the compiler to generate documentation information and include it in an XML file. Clear this check box to instruct the compiler not to create documentation.

This setting corresponds to the [/doc](#) compiler option.

Register for COM interop

Specifies whether your managed application will expose a COM object (a COM-callable wrapper) that enables a COM object to interact with the application.

By default, this check box is cleared, which specifies that the application will not allow COM interop. Select this check box to allow COM interop.

This option is not available for Windows Application or Console Application projects.

Build Events

Click this button to access the **Build Events** dialog box. Use this dialog box to specify pre-build and post-build configuration instructions for the project. This dialog box applies to Visual Basic projects only. For more information, see [Build Events Dialog Box \(Visual Basic\)](#).

Advanced Compile Options

Click this button to access the **AdvancedCompiler Settings** dialog box. Use the **AdvancedCompiler Settings** dialog box to specify a project's advanced build configuration properties. This dialog box applies to Visual Basic projects only. For more information, see [Advanced Compiler Settings Dialog Box \(Visual Basic\)](#).

See also

[How to: Specify Build Events \(Visual Basic\)](#)

[Visual Basic Command-Line Compiler](#)

[How to: Create and Edit Configurations](#)

Advanced Compiler Settings Dialog Box (Visual Basic)

1/26/2018 • 2 min to read • [Edit Online](#)

Use the **AdvancedCompiler Settings** dialog box of the **Project Designer** to specify the project's advanced build-configuration properties. This dialog box applies to Visual Basic projects only.

To access this dialog box

1. In **Solution Explorer**, choose a project node (not the **Solution** node).
2. On the **Project** menu, click **Properties**. When the **Project Designer** appears, click the **Compile** tab.
3. On the [Compile Page, Project Designer \(Visual Basic\)](#), select the **Configuration** and **Platform**. In simplified build configurations, the **Configuration** and **Platform** lists are not displayed. For more information, see [How to: Set debug and release configurations](#).
4. Click **Advanced Compile Options**.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

Optimizations

The following options specify optimizations that can in some cases make a program file smaller, make a program run faster, or speed up the build process.

Remove integer overflow checks

By default, this check box is cleared to enable integer overflow checking. Select this check box to remove integer overflow checking. If you select this check box, integer calculations might be faster. However, if you remove overflow checking and data type capacities overflow, incorrect results might be stored without an error being raised.

If overflow conditions are checked and an integer operation overflows, an [OverflowException](#) exception is thrown. If overflow conditions are not checked, integer operation overflows do not throw an exception.

Enable optimizations

By default, this check box is cleared to disable compiler optimizations. Select this check box to enable compiler optimizations. Compiler optimizations make your output file smaller, faster, and more efficient. However, because optimizations cause code rearrangement in the output file, compiler optimizations can make debugging difficult.

DLL base address

This text box displays the default DLL base address in hexadecimal format. In Class Library and Control Library projects, you can use this text box to specify the base address to be used when the DLL is created.

Generate debug info

Select **None**, **Full**, or **pdb-only** from the list. **None** specifies that no debugging information be generated. **Full** specifies that full debugging information be generated, and **pdb-only** specifies that only PDB debugging information be generated. By default, this option is set to **Full**.

Compilation Constants

Conditional compilation constants have an effect similar to that of using a `#Const` preprocessor directive in a source file, except that constants defined are public and apply to all files in the project. You can use conditional compilation constants together with the `#If...Then...#Else` directive to compile source files conditionally. See [Conditional Compilation](#).

Define DEBUG constant

By default, this check box is selected, specifying that a DEBUG constant be set.

Define TRACE constant

By default, this check box is selected, specifying that a TRACE constant be set.

Custom constants

Enter any custom constants for your application in this text box. Entries should be delimited by commas, using this form: `Name1="Value1",Name2="Value2",Name3="Value3"`.

Other Settings

Generate serialization assemblies

This setting specifies whether the compiler will create XML serialization assemblies. Serialization assemblies can improve the startup performance of `XmlSerializer` if you have used that class to serialize types in your code. By default, this option is set to **Auto**, which specifies that serialization assemblies be generated only if you have used `XmlSerializer` to encode types in your code to XML. **Off** specifies that serialization assemblies never be generated, regardless of whether your code uses `XmlSerializer`. **On** specifies that serialization assemblies always be generated. Serialization assemblies are named `TypeName.XmlSerializers.dll`.

See also

[Compile Page, Project Designer \(Visual Basic\)](#)

Build Events Dialog Box (Visual Basic)

12/22/2017 • 1 min to read • [Edit Online](#)

Use the **Build Events** dialog box to specify build configuration instructions. You can also specify the conditions under which any pre-build or post-build events are run. For more information, see [How to: Specify Build Events \(Visual Basic\)](#).

Pre-build event command line

Specifies any commands to execute before the build starts. To type long commands, click **Edit Pre-build** to display the [Pre-build Event/Post-build Event Command Line Dialog Box](#).

NOTE

Pre-build events do not run if the project is up-to-date and no build is triggered.

Post-build event command line

Specifies any commands to execute after the build ends. To type long commands, click **Edit Post-build** to display the [Pre-build Event/Post-build Event Command Line dialog box](#).

NOTE

Add a `call` statement before all post-build commands that run .bat files. For example, `call C:\MyFile.bat` or
`call C:\MyFile.bat call C:\MyFile2.bat`.

Run the post-build event

Specifies the conditions for the post-build event to run, as shown in the following table.

| OPTION | RESULT |
|--|---|
| Always | Post-build event will run, regardless of whether the build succeeds. |
| On successful build | Post-build event will run if the build succeeds. The event will run even for a project that is up-to-date, as long as the build succeeds. This is the default setting. |
| When the build updates the project output | Post-build event will run only when the compiler's output file (.exe or .dll) differs from the previous compiler output file. A post-build event is not run if a project is up-to-date. |

See Also

[Compile Page, Project Designer \(Visual Basic\)](#)

[How to: Specify Build Events \(Visual Basic\)](#)

[Pre-build Event/Post-build Event Command Line Dialog Box](#)

Debug Page, Project Designer

1/26/2018 • 2 min to read • [Edit Online](#)

WARNING

This topic does not apply to UWP apps. See [Start a debug session \(VB, C#, C++ and XAML\)](#) in the Windows Dev Center.

Use the **Debug** page of the **Project Designer** to set properties for debugging behavior in a Visual Basic or C# project.

To access the **Debug** page, select a project node in **Solution Explorer**. On the **Project** menu, choose *ProjectNameProperties*. When the **Project Designer** appears, click the **Debug** tab.

Configuration and Platform

The following options allow you to select the configuration and platform to display or modify.

Configuration

Specifies which configuration settings to display or modify. The settings can be **Debug** (default), **Release**, or **All Configurations**.

Platform

Specifies which platform settings to display or modify. The choices can include **Any CPU** (default), **x64**, and **x86**.

Start Action

Start Action indicates the item to start when the application is debugged: the project, a custom program, a URL, or nothing. By default, this option is set to **Start project**. The **Start Action** setting on the **Debug** page determines the value of the `StartAction` property.

Start project

Choose this option to specify that the executable (for Windows Application and Console Application projects) should be started when the application is debugged. This option is selected by default.

Start external program

Choose this option to specify that a specific program should be started when the application is debugged.

Start browser with URL

Choose this option to specify that a particular URL should be accessed when the application is debugged.

Start Options

Command line arguments

In this text box, enter the command-line arguments to use for debugging.

Working directory

In this text box, enter the directory from which the project will be launched. Or click the Browse button (...) to choose a directory.

Use remote machine

To debug the application from a remote computer, select this check box, and enter the path to the remote computer in the text box.

Enable Debuggers

Enable unmanaged code debugging

This option specifies whether debugging of native code is supported. Select this check box if you are making calls to COM objects or if you start a custom program written in native code that calls your project and you must debug the native code. Clear this check box to disable debugging of unmanaged code. This check box is cleared by default.

Enable SQL Server debugging

Select or clear this check box to enable or disable debugging of SQL procedures from your Visual Basic application. This check box is cleared by default.

Enable the Visual Studio hosting process

Select this check box to enable the Visual Studio hosting process. This check box is selected by default. For more information, see [Hosting Process \(vhost.exe\)](#).

To debug in a security zone, you must enable this option and **Debug this application with the selected permission set** in the [Advanced Security Settings Dialog Box](#).

See also

[Debugging in Visual Studio](#)

[Project Settings for C# Debug Configurations](#)

[Project Settings for a Visual Basic Debug Configuration](#)

[How to: Debug a ClickOnce Application with Restricted Permissions](#)

[How to: Create and Edit Configurations](#)

My Extensions Page, Project Designer (Visual Basic)

12/22/2017 • 1 min to read • [Edit Online](#)

Use the **My Extensions** page of the **Project Designer** to manage `My` namespace extensions in your project. `My` namespace extensions enable you to customize the `My` namespace to add your own custom members. For information about creating custom `My` namespace extensions, see [Extending the My Namespace in Visual Basic](#).

To access the **My Extensions** page, double-click **My Project** for your project node in **Solution Explorer**. When the **Project Designer** appears, click the **My Extensions** tab.

UIElement List

The following options enable you to add or remove `My` namespace extensions in your project. A `My` namespace extension must first be installed as a Visual Studio Item template to be available to be added. For information about publishing and installing `My` namespace extensions, see [Packaging and Deploying Custom My Extensions](#).

My namespace extensions

This list shows all the `My` namespace extensions installed in the project.

Add Extension

Click this button to add an installed `My` namespace extension to your project. A list of all possible `My` namespace extensions will appear. Select the `My` namespace extension that you want to add to your project and click **OK** to add it.

Remove Extension

Select one or more references in the **My namespace extensions** list, and then click this button to remove the `My` namespace extension from the project.

See Also

[Extending the My Namespace in Visual Basic](#)

[Packaging and Deploying Custom My Extensions](#)

[Extending the Visual Basic Application Model](#)

[Customizing Which Objects are Available in My](#)

Publish Page, Project Designer

2/11/2018 • 3 min to read • [Edit Online](#)

The **Publish** page of the **Project Designer** is used to configure properties for ClickOnce deployment.

To access the **Publish** page, select a project node in **Solution Explorer**, and then, on the **Project** menu, click **Properties**. When the **Project Designer** appears, click the **Publish** tab.

NOTE

Some of the ClickOnce properties described here can also be set in the **PublishWizard**, available from the **Build** menu or by clicking the **PublishWizard** button on this page.

UIElement List

Publishing Folder Location

Specifies the location where the application is published. Can be a drive path (`C:\deploy\myapplication`), a file share (`\server\myapplication`), or an FTP server (`ftp://ftp.microsoft.com/myapplication`). Note that text must be present in the **Publishing Location** box in order for the browse (...) button to work.

Installation Folder URL

Optional. Specifies a Web site to which users go to install the application. This is necessary only when it differs from the **Publishing Location**, for example, when the application is published to a staging server.

Install Mode and Settings

Determines whether the application is run directly from the **Publishing Location** (when **The application is available online only** is selected) or is installed and added to the **Start** menu and the **Add or Remove Programs** item in **Control Panel** (when **The application is available offline as well** is selected).

For WPF Web Browser Applications, the **The application is available offline as well** option is disabled, because such applications are available only online.

Application Files

Opens the Application Files dialog box, which is used to specify how and where individual files are installed.

Prerequisites

Opens the Prerequisites dialog box, which is used to specify prerequisite components, such as the .NET Framework, to be installed together with the application.

Updates

Opens the Application Updates dialog box, which is used to specify update behavior for the application. Not available when **The application is available online only** is selected.

Options

Opens the Publish Options dialog box, which is used to specify additional advanced publishing options.

Publish Version

Sets the publish version number for the application; when the version number is changed, the application is published as an update. Each part of the publish version (**Major**, **Minor**, **Build**, **Revision**) can have a maximum value of 65355 (**.MaxValue**), the maximum allowed by **Version**.

When you install more than one version of an application by using ClickOnce, the installation moves earlier versions of the application into a folder named Archive, in the publish location that you specify. Archiving earlier

versions in this manner keeps the installation directory clear of folders from the earlier version.

Automatically increment revision with each publish

Optional. When this option is selected (the default), the **Revision** part of the publish version number is incremented by one every time that the application is published. This causes the application to be published as an update.

Publish Wizard

Opens the Publish Wizard. Completing the Publish Wizard has the same effect as running the **Publish** command on the **Build** menu.

Publish Now

Publishes the application using the current settings. Equivalent to the **Finish** button in the **Publish Wizard**.

See Also

[Publishing ClickOnce Applications](#)

[How to: Publish a ClickOnce Application using the Publish Wizard](#)

[How to: Specify Where Visual Studio Copies the Files](#)

[How to: Specify the Location Where End Users Will Install From](#)

[How to: Specify a Link for Technical Support](#)

[How to: Specify the ClickOnce Offline or Online Install Mode](#)

[How to: Enable AutoStart for CD Installations](#)

[How to: Set the ClickOnce Publish Version](#)

[How to: Automatically Increment the ClickOnce Publish Version](#)

[How to: Specify Which Files Are Published by ClickOnce](#)

[How to: Install Prerequisites with a ClickOnce Application](#)

[How to: Manage Updates for a ClickOnce Application](#)

[How to: Change the Publish Language for a ClickOnce Application](#)

[How to: Specify a Start Menu Name for a ClickOnce Application](#)

[How to: Specify a Publish Page for a ClickOnce Application](#)

[ClickOnce Security and Deployment](#)

Prerequisites Dialog Box

3/12/2018 • 4 min to read • [Edit Online](#)

This dialog box specifies which prerequisite components are installed, how they are installed, and which order the packages are installed.

To access this dialog box, select a project node in **Solution Explorer**, and then, on the **Project** menu, click **Properties**. When the **Project Designer** appears, click the **Publish** tab. On the **Publish** page, click **Prerequisites**. For Setup projects, on the **Project** menu, click **Properties**. When the **Property Pages** dialog box appears, click **Prerequisites**.

UIElement List

| ELEMENT | DESCRIPTION |
|--|---|
| Create setup program to install prerequisite components | Includes the prerequisite components in your application's Setup program (Setup.exe) so that they will be installed before your application, in order of dependency. By default, this option is selected. If it is not selected, no Setup.exe is created. |
| Choose which prerequisites to install | <p>Specifies whether to install components such as .NET Framework, Crystal Reports, and so on.</p> <p>For example, by selecting the check box next to SQL Server 2005 Express Edition SP2, you specify that the Setup program verify whether this component is installed on the target computer and install it if it is not already installed.</p> <p>For detailed information about each prerequisite package, see the Prerequisites Information table later in this topic.</p> |
| Check Microsoft Update for more redistributable components | Clicking this link takes you to the Bootstrapper Packages to Redistribute Components website to check for updates. |
| Download prerequisites from the component vendor's web site | Specifies that the prerequisite components be installed from the vendor's Web site. This is the default option. |
| Download prerequisites from the same location as my application | Specifies that the prerequisite components be installed from the same location as the application. This copies all the prerequisite packages to the publish location. For this option to work, the prerequisite packages must be on the development computer. |
| Download prerequisites from the following location | Specifies that the prerequisite components be installed from the location that you select. You can use the Browse button to select a location. |

Prerequisites Information

The prerequisite components that appear in the **Prerequisites** dialog box might differ from those in the following list. The prerequisite packages listed in the **Prerequisites Dialog Box** are set automatically the first time that you open the dialog box. If you subsequently change the project's target framework, you will have to select the prerequisites manually to match the new target framework.

| ELEMENT | DESCRIPTION |
|---|--|
| .NET Framework 3.5 SP1 | <p>This package installs the following:</p> <ul style="list-style-type: none"> - .NET Framework versions 2.0, 3.0, and 3.5. - Support for all .NET Framework versions on 32-bit (x86) and 64-bit (x64) operating systems. - Language packs for each .NET Framework version that is installed with the package. - Service packs for .NET Framework 2.0 and 3.0. <p>.NET Framework 3.0 is included with Windows Vista, and .NET Framework 3.5 is included with Visual Studio. .NET Framework 3.5 is required for all Visual Basic and C# projects that are compiled for 32-bit operating systems and for which the target framework is set to .NET Framework 3.5, and for Visual Basic and C# projects compiled for 64-bit operating systems. (IA64 is not supported.) Note that Visual Basic and C# projects are compiled for any CPU architecture by default. For more information, see Visual Studio Multi-Targeting Overview and Deploying Prerequisites for 64-bit Applications.</p> <p>By default, this item is selected.</p> |
| Microsoft .NET Framework 4.x | <p>This package installs the .NET Framework 4.x for both the x86 and x64 platforms.</p> |
| Microsoft System CLR Types for SQL Server 2014 (x64 and x86) | <p>This package installs Microsoft System CLR Types for SQL Server 2014 for either x64 or x86.</p> |
| SQL Server 2008 R2 Express | <p>This package installs Microsoft SQL Server 2008 R2 Express, a free edition of Microsoft SQL Server 2008 R2, an ideal database for small web, server or desktop applications. It can be used for free for development and production.</p> |
| SQL Server 2012 Express | <p>This package installs Microsoft SQL Server 2012 Express.</p> |
| SQL Server 2012 Express LocalDB | <p>This package installs Microsoft SQL Server 2012 Express LocalDB.</p> |
| Visual C++ "14" Runtime Libraries (ARM) | <p>This package installs the Visual C++ run-time libraries for the Itanium architecture, which provide routines for programming for the Microsoft Windows operating system. These routines automate many common programming tasks that are not provided by the C and C++ languages.</p> <p>For more information, see C Run-Time Library Reference.</p> |
| Visual C++ "14" Runtime Libraries (x64) | <p>This package installs the Visual C++ run-time libraries for the x64 operating systems, which provide routines for programming for the Microsoft Windows operating system. These routines automate many common programming tasks that are not provided by the C and C++ languages.</p> <p>For more information, see C Run-Time Library Reference.</p> |

| ELEMENT | DESCRIPTION |
|--|---|
| Visual C++ "14" Runtime Libraries (x86) | <p>This package installs the Visual C++ run-time libraries for the x86 operating systems, which provide routines for programming for the Microsoft Windows operating system. These routines automate many common programming tasks that are not provided by the C and C++ languages.</p> <p>For more information, see C Run-Time Library Reference.</p> |

See also

- [Publish Page, Project Designer](#)
- [Application Deployment Prerequisites](#)
- [Deploying Prerequisites for 64-bit Applications](#)
- [Visual Studio Multi-Targeting Overview](#)

References Page, Project Designer (Visual Basic)

12/22/2017 • 2 min to read • [Edit Online](#)

Use the **References** page of the **Project Designer** to manage references, Web references, and imported namespaces in your project. Projects can contain references to COM components, XML Web services, .NET Framework class libraries or assemblies, or other class libraries. For more information on using references, see [Managing references in a project](#).

To access the **References** page, choose a project node (not the **Solution** node) in **Solution Explorer**. Then choose **Project, Properties** on the menu bar. When the Project Designer appears, click the **References** tab.

UIElement List

The following options allow you to select or remove references and imported namespaces in your project.

Unused References

Click this button to access the **Unused References** dialog box.

The **Unused References** dialog box allows you to remove references that are included in your project but not actually used by the code. It contains a grid that lists the **Reference Name**, the **Path**, and other information about the unused namespace references in your project. In the grid, select the namespace references that you want to remove from your project and click **Remove**.

Reference Paths

Click this button to access the **Reference Paths** dialog box.

NOTE

When the project system finds an assembly reference, the system resolves the reference by looking in the following locations, in the following order:

1. The project folder. The project folder files appear in **Solution Explorer** when **Show All Files** isn't in effect.
 - a. Folders that are specified in the **Reference Paths** dialog box.
 - b. Folders that display files in the **Add Reference** dialog box.
- c. The project's obj folder. (When you add a COM reference to your project, one or more assemblies may be added to the project's obj folder.)

References

This list shows all references in the project, used or unused.

Add

Click this button to add a reference or Web reference to the **References** list.

Choose **Reference** to add a reference to your project using the Add Reference dialog box.

Choose **Web Reference** to add a Web reference to your project using the Add Web Reference dialog box.

Remove

Select one or more references in the **References** list, then click this button to delete it.

Update Web Reference

Select a Web reference in the **References** list and click this button to update it.

Imported namespaces

You can type your own namespace in this box and click **Add User Import** to add it to the namespaces list.

You can create aliases for user-imported namespaces. To do this, enter the alias and the namespace in the format *alias=namespace*. This is useful if you are using long namespaces, for example:

```
Http= MyOrg.ObjectLib.Internet.WebRequestMethods.Http .
```

Add User Import

Click this button to add the namespace specified in the **Imported namespaces** box to the list of imported namespaces. The button is active only if the specified namespace is not already in the list.

Namespaces list

This list shows all available namespaces. The check boxes for namespaces included in your project are selected.

Update User Import

Select a user-specified namespace in the namespaces list, type the name that you want to replace it with in the **Imported namespaces** box, and then click this button to change to the new namespace. The button is active only if the selected namespace is one that you added to the list by using the **Add User Import** button. You can add:

- Classes or namespaces, such as `System.Math`.
- Aliased imports, such as `VB=Microsoft.VisualBasic`.
- XML namespaces, such as `<xmllns:xsl="http://www.w3.org/1999/XSL/Transform">`.

See Also

[Managing references in a project](#)

[How to: Add or Remove Imported Namespaces \(Visual Basic\)](#)

[Imports Statement \(XML Namespace\)](#)

Security Page, Project Designer

12/22/2017 • 2 min to read • [Edit Online](#)

The **Security** page of the **Project Designer** is used to configure code access security settings for applications that are deployed by using ClickOnce deployment. For more information, see [Code Access Security for ClickOnce Applications](#).

To access the **Security** page, click a project node in **Solution Explorer**, and then, on the **Project** menu, click **Properties**. When the **Project Designer** appears, click the **Security** tab.

Security Settings

Enable ClickOnce Security Settings

Determines whether security settings are enabled at design time. When this option is cleared, all other options on the **Security** page are unavailable.

NOTE

When you publish an application by using the **Publish** wizard, this option is automatically enabled.

When you select this option, you have the choice of selecting one of two radio buttons: **This is a full trust application** or **This is a partial trust application**.

By default, for WPF Web Browser Application projects, this option is selected.

By default, for all other project types, this option is cleared.

This is a full trust application

If you select this option, the application requests Full Trust permissions when it is installed or run on a client computer. Avoid using Full Trust if possible, because your application will be granted unrestricted access to resources such as the file system and the registry.

By default, for WPF Web Browser Application projects, this option is set to Partial Trust.

By default, for all other project types, this option is set to Full Trust.

This is a partial trust application

If you select this option, the application requests Partial Trust permissions when it is installed or run on a client computer. *Partial Trust* means that only the actions that are permitted under the requested code access security permissions will run. For more information about how to configure security permissions, see [Code Access Security for ClickOnce Applications](#).

You can specify the Partial Trust security settings by configuring the options in the **ClickOnce Security Permissions** area.

ClickOnce Security Permissions

Zone your application will be installed from

Specifies a default set of code-access security permissions. Choose **Internet** or **Local Intranet** for a restricted permission set, or choose **(Custom)** to configure a custom permission set. If the application requests more permissions than granted in a zone, a ClickOnce trust prompt appears for the end user to grant the additional permissions. For more information about how to configure security permissions, see [Code Access Security for ClickOnce Applications](#).

[ClickOnce Applications](#)

By default, for WPF Web Browser Application projects, this option is set to **Internet**.

Edit Permissions XML

Opens the application manifest template (app.manifest) to configure the permissions for the **(Custom)** permission set.

Advanced

Opens the [Advanced Security Settings Dialog Box](#), which is used to configure settings for debugging the application with restricted permissions. These settings are checked during debugging, and permission exceptions indicate that your application may need more permissions than defined in a zone.

See Also

[WebBrowserPermission](#)

[MediaPermission](#)

[Code Access Security for ClickOnce Applications](#)

[How to: Enable ClickOnce Security Settings](#)

[How to: Set a Security Zone for a ClickOnce Application](#)

[How to: Set Custom Permissions for a ClickOnce Application](#)

[How to: Debug a ClickOnce Application with Restricted Permissions](#)

[ClickOnce Security and Deployment](#)

[Project Properties Reference](#)

[Advanced Security Settings Dialog Box](#)

Advanced Security Settings Dialog Box

12/22/2017 • 1 min to read • [Edit Online](#)

This dialog box allows you to specify security settings related to debugging in zone.

To access this dialog box, select a project node in **Solution Explorer**, and then, on the **Project** menu, click **Properties**. When the **Project Designer** appears, click the **Security** tab. On the **Security** page, select **Enable ClickOnce Security Settings**, click **This is a partial trust application**, and then click **Advanced**.

UIElement List

Debug this application with the selected permission set

If you select this check box, the permission set selected on the **Security** page is used during debugging. By default, this option is selected.

For debugging in a security zone to work, this option must be enabled; also, the **Enable the Visual Studio hosting process** option (available on the **Debug** page of the **Project Designer**) must be enabled.

For WPF Web Browser Application projects, the **Debug this application with the selected permission set** option is checked and disabled.

Grant the application access to its site of origin

If you select this check box, the application can access the Web site or server share on which it is published. By default, this option is selected.

Debug this application as if it were downloaded from the following URL

If you have to allow the application to access the Web site or server share corresponding to the **Installation URL** you specified on the **Publish** page, type that URL here. This option is available only when **Grant the application access to its site of origin** is selected.

See Also

[Security Page, Project Designer](#)

Services Page, Project Designer

1/22/2018 • 2 min to read • [Edit Online](#)

Client application services provide simplified access to Microsoft Ajax login, roles, and profile services from Windows Forms and Windows Presentation Foundation (WPF) applications. You can use the **Services** page of the **Project Designer** to enable and configure client application services for your project.

With client application services, you can use a centralized server to authenticate users, determine each user's assigned role or roles, and store per-user application settings that you can share across the network. For more information, see [Client Application Services](#).

To access the **Services** page, select a project node in **Solution Explorer**, and then click **Properties** on the **Project** menu. When the **Project Designer** appears, click the **Services** tab.

Task List

[How to: Configure Client Application Services](#)

UIElement List

Configuration

This control is not editable on this page. For a description of this control, see [Compile Page, Project Designer \(Visual Basic\)](#) or [Build Page, Project Designer \(C#\)](#).

Platform

This control is not editable on this page. For a description of this control, see [Compile Page, Project Designer \(Visual Basic\)](#) or [Build Page, Project Designer \(C#\)](#).

Enable client application services

Select to enable client application services. You must specify service locations on the **Services** page to use client application services.

Use Windows authentication

Indicates that the authentication provider will use Windows-based authentication, that is, the identity supplied by the Windows operating system.

Use Forms authentication

Indicates that the authentication provider will use forms authentication. This means that your application must provide a user interface for login. For more information, see [How to: Implement User Login with Client Application Services](#).

Authentication service location

Used only with forms authentication. Specifies the location of the authentication service.

Optional: Credentials provider

Used only with forms authentication. Indicates the `IClientFormsAuthenticationCredentialsProvider` implementation that the authentication service will use to display a login dialog box when your application calls the `static System.Web.Security.Membership.ValidateUser` method and passes empty strings or `null` for the parameters. If you leave this box blank, you must pass a valid user name and password to the `System.Web.Security.Membership.ValidateUser` method. You must specify the credentials provider as an assembly-qualified type name. For more information, see `System.Type.AssemblyQualifiedName` and [Assembly Names](#). In its simplest form, an assembly-qualified type name looks similar to the following example:

MyNamespace.MyLoginClass, MyAssembly

Roles service location

Specifies the location of the roles service.

Web settings service location

Specifies the location of the profile (Web settings) service.

Advanced

Opens the [Advanced Settings for Services Dialog Box](#), which you can use to override default behavior. For example, you can use this dialog box to specify a database for offline storage instead of using the local file system. For more information, see [Advanced Settings for Services Dialog Box](#).

See also

[Client Application Services](#)

[Advanced Settings for Services Dialog Box](#)

[How to: Configure Client Application Services](#)

[Compile Page, Project Designer \(Visual Basic\)](#)

[Build Page, Project Designer \(C#\)](#)

Advanced Settings for Services Dialog Box

1/26/2018 • 2 min to read • [Edit Online](#)

Client application services provide simplified access to Microsoft Ajax login, roles, and profile services from Windows Forms and Windows Presentation Foundation (WPF) applications. You can use the **Services** page in the **Project Designer** to configure client application services. For more information about the **Services** page, see [Services Page, Project Designer](#).

Use the **Advanced Settings for Services** dialog box of the **Services** page in the **Project Designer** to configure advanced settings for client application services. By using these settings, you can override some default application service behaviors to enable less common scenarios. For more information, see [Client Application Services](#).

To access the **Advanced Settings for Services** dialog box, select a project node in **Solution Explorer**, and then click **Properties** on the **Project** menu. When the **Project Designer** appears, click the **Services** tab, and then click the **Advanced** button. This button will be disabled until you enable client application services.

Task List

- [How to: Configure Client Application Services](#)

UIElement List

Save password hash locally to enable offline login

Specifies whether an encrypted form of the user's password will be cached locally to enable the user to log in when the application is in offline mode. This option is selected by default.

Require users to log on again whenever the server cookie expires

Specifies whether previously authenticated users are automatically reauthenticated when your application accesses the roles or profile service and the server authentication cookie has expired. Select this option to deny access to the application services and require explicit reauthentication after the cookie expires. This is useful for applications deployed in public locations to make sure that users who leave the application running after use will not remain authenticated indefinitely. This option is cleared by default.

Role service cache timeout

Specifies the amount of time the client role provider will use cached role values instead of accessing the roles service. Set this time interval to a small value when roles are updated frequently or to a larger value when roles are updated infrequently. The default value is one day.

The role provider accesses the cached role values or the roles service when you call the [IsInRole](#) method. To programmatically clear the cache and force this method to access the remote service, call the [ResetCache](#) method.

Use custom connection string

Specifies whether the client service providers will use a custom data store for the local cache. By default, the service providers will use the local file system for the cache. Selecting this option will automatically populate the text box with a default connection string. You can keep the default connection string to automatically generate and use a SQL Server Compact Edition database, or you can specify a connection string to an existing SQL Server database. For more information, see [How to: Configure Client Application Services](#). This option is cleared by default.

See also

[Client Application Services](#)

[Services Page, Project Designer](#)

How to: Configure Client Application Services

Signing Page, Project Designer

1/26/2018 • 4 min to read • [Edit Online](#)

Use the **Signing** page of the **Project Designer** to sign the application and deployment manifests and also to sign the assembly (strong-name signing).

Notice that the signing of application and deployment manifests is a process distinct from the signing of an assembly, although both tasks are performed on the **Signing** page.

Also, the storage of key-file information differs for manifest signing and assembly signing. For manifest signing, key information is stored in your computer's cryptographic storage database and the current user's Windows certificate store. For assembly signing, key information is stored only in your computer's cryptographic storage database.

To access the **Signing** page, select a project node in **Solution Explorer**, and then, on the **Project** menu, click **Properties**. When the **Project Designer** appears, click the **Signing** tab.

Application and Deployment Manifest Signing

Sign the ClickOnce manifests check box

Select this check box to sign the application and deployment manifests with a public/private key pair. For more information about how to do this, see [How to: Sign Application and Deployment Manifests](#).

Select from Store button

Allows you to select an existing certificate from the current user's personal certificate store. You can select one of these certificates to sign your application and deployment manifests.

Clicking **Select from Store** opens the **Select a Certificate** dialog box, which lists certificates in your personal certificate store that are currently valid (not expired) and that have private keys. The purpose of the certificate you select should include code signing.

If you click **view certificate properties**, the **Certificate Details** dialog box appears. This dialog box includes detailed information about the certificate, and includes additional options. You can click **Learn more about certificates** to view additional Help information.

Select from File button

Allows you to select a certificate from an existing key file.

Clicking **Select from File** opens the **Select File** dialog box, which enables you to select a certificate key (.pfx) file. The file must be password protected and cannot already be located in your personal certificate store.

In the **Enter password to open file** dialog box, enter a password to open the certificate key (.pfx) file. The password information is stored in your personal key container list and your personal certificate store.

Create Test Certificate button

Allows you to create a certificate for testing. The test certificate is used for signing your ClickOnce application and deployment manifests.

Clicking **Create Test Certificate** opens the **Create Test Certificate** dialog box, in which you can type a password for the strong-name key file for the test certificate. The file is named *projectname_TemporaryKey.pfx*. If you click **OK** without typing a password, the .pfx file is not password encrypted.

Timestamp server URL box

Specifies the address of a server that timestamps your signature. When you provide a certificate, this external site

verifies the time that the application was signed.

Assembly Signing

Sign the assembly check box

Select this check box to sign the assembly and create a strongly named key file. For more information about signing the assembly by using the **Project Designer**, see [How to: Sign an Assembly \(Visual Studio\)](#).

This option uses the Al.exe tool provided by the `![INCLUDEwinsdklong]`.

Choose a strong name key file list

Enables you to specify a new or existing strongly named key file that is used to sign the assembly. Select **<Browse...>** to select an existing key file.

Select **<New...>** to create a new key file with which to sign the assembly. The **Create Strong Name Key** dialog box appears, which you can use to specify a key file name and protect the key file with a password. The password must be at least 6 characters long. If you specify a password, a Personal Information Exchange (.pfx) file is created; if you do not specify a password, a strongly named key (.snk) file is created.

Change Password button

Changes the password for the Personal Information Exchange (.pfx) key file that is used to sign the assembly.

Clicking **Change Password** opens the **Change Key Password** dialog box. In this dialog box, **Old password** is the current password for the key file. **New password** must be at least 6 characters long. The password information is stored in current user's Windows certificate store.

Delay sign only check box

Select this check box to enable delay signing.

Note that a delay signed project will not run and cannot be debugged. You can, however, use the [Sn.exe \(Strong Name Tool\)](#) with the `-vr` option to skip verification during development.

NOTE

When you sign an assembly, you might not always have access to a private key. For example, an organization might have a closely guarded key pair that developers don't have access to on a daily basis. The public key might be available, but access to the private key is restricted to a few individuals. In such a case, you can use *delayed* or *partial signing* to provide the public key, deferring the addition of the private key until the assembly is handed off.

See also

[Project Properties Reference](#)

[Managing Assembly and Manifest Signing](#)

[How to: Sign Application and Deployment Manifests](#)

[How to: Sign an Assembly \(Visual Studio\)](#)

[How to: Sign an Assembly with a Strong Name](#)

[Strong-Named Assemblies](#)

Property Pages, JavaScript

1/22/2018 • 3 min to read • [Edit Online](#)

The **Property Pages** provides access to project settings. You can use the pages that appear in the **Property Pages** to change project properties.

To access the project properties, select a project node in **Solution Explorer**. On the **Project** menu, click **Properties**.

NOTE

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

The following pages and options appear in the **Property Pages**.

Configuration and Platform Page

Use the following options to select the configuration and platform to display or modify.

Configuration

Specifies the configuration settings to display or modify. The settings are **Debug** (default), **Release**, **All Configurations**, or a user-defined configuration. For more information, see [How to: Set debug and release configurations in Visual Studio](#).

Platform

Specifies the platform settings to display or modify. The settings are **Any CPU** (default for Windows 8.x Store apps), **x64**, **ARM**, **x86**, or a user-defined platform. For more information, see [How to: Set debug and release configurations in Visual Studio](#).

General Page

Use the following options to set general properties of the project.

NOTE

Some options are only available in UWP apps.

Output Path

Specifies the location of the output files for the project's configuration. The path is relative; if you enter an absolute path, the absolute path is saved in the project. The default path is bin\Debug.

When you use simplified build configurations, the project system determines whether to build a debug or release version. When you click **Debug, Start Debugging** (or press F5) the build is put in the debug location regardless of the **Output path** you specify. However, the **Build Solution** command on the **Build** menu puts it in the location you specify. To enable advanced build configurations, on the menu bar, choose **Tools, Options**. In the **Options** dialog box, expand **Projects and Solutions**, select **General**, and then clear the **Show advanced build configurations** check box. This gives you manual control over all configuration values and whether a debug or release version is built.

Default Language

Specifies the default language for the project. The language option selected in **Clock, Language, and Region** in Control Panel specifies the user's preferred language. By specifying a default language for the project, you make sure that the specified default language resources are used if the user's preferred language does not match the language resources provided in the application.

Debug Page

Use the following options to set properties for debugging behavior in the project.

NOTE

Some options are only available in UWP apps.

Debugger to Launch

Specifies the default host for the debugger.

- Select **Local Machine** to start the application on the Visual Studio host computer. For more information, see [Running apps on the local machine](#).
- Select **Simulator** to start the application in the Simulator. For more information, see [Running apps in the simulator](#).
- Select **Remote Machine** to start the application on a remote computer. For more information about remote debugging, see [Running apps on a remote machine](#).

Launch Application

Specifies whether to start the application when you press F5 or click **Debug, Start Debugging**. Select **Yes** to start the application; otherwise, select **No**. If you select **No**, you can still debug the application if you use a different method to start it.

Debugger Type

Specifies the types of code to debug. Select **Script Only** to debug JavaScript code. Select **Managed Only** to debug code that is managed by the common language runtime. Select **Native Only** to debug C++ code. Select **Native with Script** to debug C++ and JavaScript. Select **Mixed (Managed and Native)** to debug both managed and C++ code.

Allow Local Network Loopback

Specifies whether access to the IP loopback address is allowed for app testing. Select **Yes** to allow use of the loopback address if the client app is on the same machine where the server application is running; otherwise, select **No**. This property is available only if the **Debugger to Launch** property is set to **Remote Machine**.

Machine Name

Specifies the name of the remote computer to host the debugger. This property is available only if **Debugger to Launch** is set to **Remote Machine**.

Require Authentication

Specifies whether the remote computer requires authentication. This property is available only if **Debugger to Launch** is set to **Remote Machine**.

Properties Window

12/22/2017 • 2 min to read • [Edit Online](#)

Use this window to view and change the design-time properties and events of selected objects that are located in editors and designers. You can also use the **Properties** window to edit and view file, project, and solution properties. You can find **Properties** Window on the **View** menu. You can also open it by pressing F4 or by typing **Properties** in the **Quick Launch** window.

The **Properties** window displays different types of editing fields, depending on the needs of a particular property. These edit fields include edit boxes, drop-down lists, and links to custom editor dialog boxes. Properties shown in gray are read-only.

UIElement List

Object name

Lists the currently selected object or objects. Only objects from the active editor or designer are visible. When you select multiple objects, only properties common to all selected objects appear.

Categorized

Lists all properties and property values for the selected object, by category. You can collapse a category to reduce the number of visible properties. When you expand or collapse a category, you see a plus (+) or minus (-) to the left of the category name. Categories are listed alphabetically.

Alphabetical

Alphabetically sorts all design-time properties and events for selected objects. To edit an undimmed property, click in the cell to its right and enter changes.

Property Pages

Displays the **Property Pages** dialog box or **Project Designer** for the selected item. Property Pages displays a subset, the same or a superset of the properties available in the **Properties** window. Use this button to view and edit properties related to your project's active configuration.

Properties

Displays the properties for an object. Many objects also have events that can be viewed using the **Properties** window.

Sort by Property Source

Groups properties by source, such as inheritance, applied styles, and bindings. Only available when editing XAML files in the designer.

Events

Displays the events for an object.

NOTE

This **Properties** window toolbar control is only available when a form or control designer is active in the context of a Visual C# project. When editing XAML files, events appear on a separate tab of the properties window.

Messages

Lists all Windows messages. Allows you to add or delete specified handler functions for the messages provided for the selected class.

NOTE

This **Properties** window toolbar control is only available when **Class View** is the active window in the context of a Visual C++ project.

Overrides

Lists all virtual functions for the selected class and allows you to add or delete overriding functions.

NOTE

This **Properties** window toolbar control is only available when **Class View** is the active window in the context of a Visual C++ project.

Description pane

Shows the property type and a short description of the property. You can turn the description of the property off and on using the Description command on the shortcut menu.

NOTE

This **Properties** window toolbar control is not available when editing XAML files in the designer.

Thumbnail view

Shows a visual representation of the currently selected element when editing XAML files in the designer.

Search

Provides a Search function for properties and events when editing XAML files in the designer. The search box responds to partial word searches and updates search results as you type.

See Also

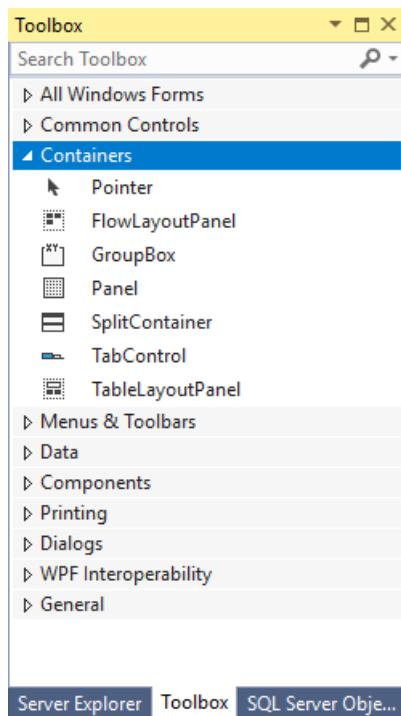
[Project Properties Reference](#)

[Customizing window layouts](#)

Toolbox

1/22/2018 • 2 min to read • [Edit Online](#)

The **Toolbox** window displays controls that you can add to Visual Studio projects. To open Toolbox, choose **Toolbox** on the **View** menu.



You can drag and drop different controls onto the surface of the designer you are using, and resize and position the controls.

Toolbox appears in conjunction with designer views, such as the designer view of a XAML file. Toolbox displays only those controls that can be used in the current designer. You can search within Toolbox to further filter the items that appear.

NOTE

For some project types, Toolbox may not show any items.

The .NET Framework version that your project targets also affects the set of controls visible in Toolbox. You can set your project to target a different version of the .NET Framework from the project's property pages. Select the project node in **Solution Explorer**, and then on the menu bar, choose **Project > <project> Properties**. On the **Application** tab, use the **Target framework** drop-down.

Managing the Toolbox window and its controls

By default Toolbox is collapsed along the left side of the Visual Studio IDE, and appears when the cursor is moved over it. You can pin Toolbox (by clicking the **Pin** icon on its toolbar) so that it remains open when you move the cursor. You can also undock the Toolbox window and drag it anywhere on your screen. You can dock, undock, and hide Toolbox by right-clicking its toolbar and selecting one of the options.

You can rearrange the items in a Toolbox tab or add custom tabs and items by using the following commands on the context menu:

- **Rename item** - Renames the selected item.
- **Show All** - Shows all possible controls (not just the ones that apply to the current designer).
- **List View** - Shows the controls in a vertical list. If unchecked, the controls appear horizontally.
- **Choose Items** - Opens the **Choose Toolbox Items** dialog box so that you can specify the items that appear in the **Toolbox**. You can show or hide an item by selecting or clearing its check box.
- **Sort items alphabetically** - Sorts the items by name.
- **Reset Toolbar** - Restores the default Toolbox settings and items.
- **Add Tab** - Adds a new Toolbox tab.
- **Move Up** - Moves the selected item up.
- **Move Down** - Moves the selected item down.

Creating and distributing custom Toolbox controls

You can create custom Toolbox controls, starting either with a project template that's based on [Windows Presentation Foundation](#) or on [Windows Forms](#). You can then distribute your custom control to your teammates, or publish it on the web by using the [Toolbox Controls Installer](#).

Help on Toolbox tabs

The following topics provide more information about some of the available **Toolbox** tabs.

- [Toolbox, Data Tab](#)
- [Toolbox, Components Tab](#)
- [Toolbox, HTML Tab](#)

Toolbox, Components Tab

1/30/2018 • 2 min to read • [Edit Online](#)

Displays components you can add to Visual Basic and C# designers. In addition to the .NET Framework components that are included with Visual Studio, such as the [MessageQueue](#) and [EventLog](#) components, you can add your own or third-party components to this tab.

To display this tab, from the **View** menu, select **Toolbox**. In the **Toolbox**, select the **Components** tab.

BackgroundWorker

Creates a `System.ComponentModel.BackgroundWorker` component instance that can run an operation on a separate, dedicated thread.

DirectoryEntry

Creates a [DirectoryEntry](#) component instance that encapsulates a node or object in the Active Directory hierarchy and can be used to interact with Active Directory service providers.

DirectorySearcher

Creates a [DirectorySearcher](#) component instance that you can use to perform queries against the Active Directory.

ErrorProvider

Creates a `System.Windows.Forms.ErrorProvider` component instance, which indicates to the end user that a control on a form has an error associated with it.

EventLog

Creates an [EventLog](#) component instance you can use to interact with system and custom event logs, including writing events to a log and reading log data.

FileSystemWatcher

Creates a [FileSystemWatcher](#) component instance that you can use to monitor for changes to any directory or file to which you have access.

HelpProvider

Creates a `System.Windows.Forms.HelpProvider` component instance that provides pop-up or online Help for controls.

ImageList

Creates a `System.Windows.Forms.ImageList` component instance that provides methods to manage a collection of `System.Drawing.Image` objects.

MessageQueue

Creates a [MessageQueue](#) component instance that you can use to interact with message queues, including reading messages from and writing messages to queues, processing transactions, and performing queue administration tasks.

PerformanceCounter

Creates a [PerformanceCounter](#) component instance that you can use to interact with Windows performance counters, including creating new categories and instances, reading values from counters, and performing calculations on counter data.

Process

Creates a [Process](#) component instance you can use to stop, start, and manipulate the data associated with processes on your system.

SerialPort

Creates a `System.IO.Ports.SerialPort` component instance that provides synchronous and event-driven I/O, access to pin and break states, and access to serial driver properties.

ServiceController

Creates a [ServiceController](#) component instance you can use to manipulate existing services, including starting and stopping services and sending commands to them.

Timer

Creates a [Timer](#) component instance you can use to add time-based functionality to your Windows-based applications. For more information, see [Timer Component](#).

NOTE

There is also a system-based [Timer](#) that you can add to the [Toolbox](#). This [Timer](#) is optimized for server applications, and the Windows Forms [Timer](#) is best suited for use on Windows Forms.

See also

[Programming with Components](#)

[Component Programming Walkthroughs](#)

[Toolbox](#)

Toolbox, Data Tab

12/22/2017 • 1 min to read • [Edit Online](#)

Displays data objects you can add to a forms and components. The **Data** tab of the **Toolbox** appears when you create a project that has an associated designer. The **Toolbox** appears by default in the Visual Studio integrated development environment; if you need to display the **Toolbox**, select **Toolbox** from the **View** menu.

TIP

Running the Data Source Configuration Wizard will automatically create and configure most data items. For more information, see [Add new data sources](#).

UI Element List

To go directly to the .NET Framework reference page for a component, press **F1** on the item in the **Toolbox** or on the component item in the tray of the designer.

| NAME | DESCRIPTION |
|----------------------------------|---|
| DataSet | Adds an instance of a typed or untyped dataset to the form or component. When you drag this object onto a designer, it displays a dialog box that allows you to select an existing typed dataset class or specify that you want to create a new, blank, untyped dataset. Note: You do not use the DataSet object on the Toolbox to create a new typed dataset schema and class. For more information, see Create and configure datasets . |
| DataGridView | Provides a powerful and flexible way to display data in a tabular format. |
| BindingSource | Simplifies the process of binding controls to an underlying data source. |
| BindingNavigator | Represents the navigation and manipulation user interface (UI) for controls on a form that are bound to data. |

See Also

- [Accessing Data in Visual Studio](#)
- [Visual Studio data tools for .NET](#)
- [Dataset tools in Visual Studio](#)
- [Bind controls to data in Visual Studio](#)
- [Bind Windows Forms controls to data in Visual Studio](#)
- [Edit data in datasets](#)
- [Validate data in datasets](#)
- [Saving Data](#)

Toolbox, HTML tab

1/22/2018 • 5 min to read • [Edit Online](#)

The **HTML** tab of the Toolbox provides components that are useful on Web pages and Web forms. To view this tab, first open a document for editing in the HTML designer. On the **View** menu, click **Toolbox**, and then click the **HTML** tab of the Toolbox.

To create an instance of a tool on the **HTML** tab, either double-click the tool to add it to your document at the current insertion point, or select the tool and drag it to the desired position on the editing surface.

UI elements

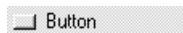
The following tools are available by default on the HTML tab.

Pointer



This tool is selected by default when any Toolbox tab opens. It cannot be deleted. The pointer enables you to drag objects onto the Design view surface, resize them, and reposition them on the page or form. For more information, see [Toolbox](#).

Input (Button)

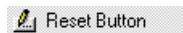


Inserts an `input` element of `type="button"`. To change the text that is displayed, edit the `name` property. By default, `id="Button1"` is inserted for the first button, `id="Button2"` for the second, and so on.

When you drag **Input (Button)** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<input id="Button1" type="button" value="Button" name="Button1">
```

Input (Reset)



Inserts an `input` element of `type="reset"`. To change the text that is displayed, edit the `name` property. By default, `id="Reset1"` is inserted for the first reset button, `id="Reset2"` for the second, and so on.

When you drag **Input (Reset)** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<input id="Reset1" type="reset" value="Reset" name="Reset1">
```

Input (Submit)



Inserts an `input` element of `type="submit"`. To change the text that is displayed, edit the `name` property. By default, `id="Submit1"` is inserted for the first submit button, `id="Submit2"` for the second, and so on.

When you drag **Input (Submit)** onto the Design view surface, HTML markup like the following is inserted into

your document:

```
<input id="Submit1" type="submit" value="Submit" name="Submit1">
```

Input (Text)



Inserts an `input` element of `type="text"` in your document. To change the default text that is displayed, edit the `value` attribute. By default, `id="Text1"` is inserted for the first text field, `id="Text2"` for the second, and so on.

When you drag **Input (Text)** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<input id="Text1" TYPE="text" value="Text Field" name="Text1">
```

IMPORTANT

It is recommended that you validate all user input. For more information, see [Validating User Input in ASP.NET Web Pages \(Razor\) Sites](#).

Input (File)



Inserts an `input` element of `type="file"` in your document. By default, `id="File1"` is inserted for the first file field, `id="File2"` for the second, and so on.

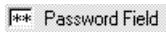
When you drag **Input (File)** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<input id="File1" type="file" name="File1">
```

IMPORTANT

It is recommended that you validate all user input. For more information, see [Validating User Input in ASP.NET Web Pages \(Razor\) Sites](#).

Input (Password)



Inserts an `input` element of `type="password"`. By default, `id="Password1"` is inserted for the first password field, `id="Password2"` for the second, and so on.

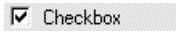
When you drag **Input (Password)** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<input id="Password1" type="password" name="Password1">
```

IMPORTANT

If your application transmits user names and passwords, you should configure your Web site to use Secure Sockets Layer (SSL) to encrypt the transmission. For more information, see "Securing Connections with SSL" in the [IIS Operations Guide](#). Additionally, it is recommended that you validate all user input. For more information, see [Validating User Input in ASP.NET Web Pages \(Razor\) Sites](#).

Input (Check box)



Inserts an `input` element of `type="checkbox"`. To change the text that is displayed, edit the `name` property. By default, `id="Checkbox1"` is inserted for the first check box, `id="Checkbox2"` for the second, and so on.

When you drag **Input (Check box)** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<input id="Checkbox1" type="checkbox" name="Checkbox1">
```

Input (Radio)



Inserts an `input` element of `type="radio"`. To change the text that is displayed, edit the `name` property. By default, `id="Radio1"` is inserted for the first radio button, `id="Radio2"` for the second, and so on.

When you drag **Input (Radio)** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<input id="Radio1" type="radio" name="Radio1">
```

Input (Hidden)

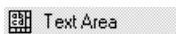


Inserts an `input` element of `type="hidden"`. By default, `id="Hidden1"` is inserted for the first hidden field, `id="Hidden2"` for the second, and so on.

When you drag **Input (Hidden)** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<input id="Hidden1" type="hidden" name="Hidden1">
```

Textarea



Inserts a `textarea` element. You can resize the text area, or use its scroll bars to view text that extends beyond its display area. To change the default text that is displayed, edit the `value` attribute. By default, `id="textarea1"` is inserted for the first text area, `id="textarea 2"` for the second, and so on.

When you drag **Textarea** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<textarea id=" textarea 1 name=" textarea 1" rows=2 cols=20></textarea>
```

IMPORTANT

It is recommended that you validate all user input. For more information, see [Validating User Input in ASP.NET Web Pages \(Razor\) Sites](#).

Table



Inserts a `table` element.

When you drag **Table** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<table cellspacing="1" width="75%" border=1> <tr><td></td></tr></table>
```

Image



Inserts an `img` element. Edit this element to specify its `src` and its `alt` text.

When you drag **Image** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<img alt="" src="">
```

Select



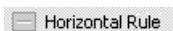
Inserts a dropdown `select` element (without a `size` attribute). By default, `id="select1"` is inserted for the first list box, `id="select2"` for the second, and so on.

When you drag **Select** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<select id="select1" name="select1"><option selected></option></select>
```

You can create a multi-line `select` element by increasing the value of the `size` property.

Horizontal Rule



Inserts an `hr` element. To increase the thickness of the line, edit the `size` attribute.

When you drag **Horizontal Rule** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<hr width="100%" size=1>
```

Div



Inserts a `div` element that includes an `ms_positioning="FlowLayout"` attribute. Except for the width and height, this item is identical to a Flow Layout Panel. To format the text that is contained within the `div` element, add a `class="stylename"` attribute to the opening tag.

When you drag **Div** onto the Design view surface, HTML markup like the following is inserted into your document:

```
<div ms_positioning="FlowLayout" style="width: 70px; position: relative; height: 15px">Label</div>
```

See also

[Toolbox](#)

Devenv command line switches

3/2/2018 • 4 min to read • [Edit Online](#)

Devenv lets you set various options for the integrated development environment (IDE), as well as build, debug, and deploy projects, from the command line. Use these switches to run the IDE from a script or a .bat file, for example a nightly build script, or to start the IDE in a particular configuration.

NOTE

For build-related tasks, it is recommended that you use MSBuild instead of devenv. For more information, see [MSBuild command-line reference](#).

Devenv switch syntax

By default, devenv commands pass switches to the devenv.com utility. The devenv.com utility delivers output through standard system streams, such as `stdout` and `stderr`. The utility determines the appropriate I/O redirection when it captures output, for example to a .txt file.

On the other hand, commands that begin with `devenv.exe` can use the same switches, but the devenv.com utility is bypassed.

The syntax rules for `devenv` switches resemble those for other DOS command-line utilities. The following syntax rules apply to all `devenv` switches and their arguments:

- Commands begin with `devenv`.
- Switches are not case-sensitive.
- When specifying a solution or project, the first argument is the name of the solution file or project file, including file path.
- If the first argument is a file that is not a solution or project, that file opens in the appropriate editor, in a new instance of the IDE.
- When you supply a project file name instead of a solution file name, a `devenv` command searches the parent folder of the project file for a solution file that has the same name. For example, the command `devenv /build myproject1.vbproj` searches the parent folder for a solution file that is named "myproject1.sln".

NOTE

One and only one solution file that references this project should be located in its parent folder. If the parent folder contains no solution file that references this project, or if the parent folder contains two or more solution files that reference it, then a temporary solution file is created.

- When file paths and file names include spaces, you must enclose them in quotation marks (""). For example, "c:\project a\".
- Insert one space character between switches and arguments on the same line. For example, the command `devenv /log output.txt` opens the IDE and outputs all log information for that session to output.txt.

- You cannot use pattern-matching syntax in `devenv` commands.

Devenv switches

The following command-line switches display the IDE and perform the described task.

| COMMAND LINE SWITCH | DESCRIPTION |
|--------------------------------|---|
| /Command | Starts the IDE and executes the specified command. |
| /DebugExe | Loads a C++ executable under the control of the debugger. This switch is not available for Visual Basic or C# executables. For more information, see Automatically start a process in the debugger . |
| /LCID or /l | Sets the default language for the IDE. If the specified language is not included in your installation of Visual Studio, this setting is ignored. |
| /Log | Starts Visual Studio and logs all activity to the log file. |
| /Run or /r | Compiles and runs the specified solution. |
| /Runexit | Compiles and runs the specified solution, minimizes the IDE when the solution is run, and closes the IDE after the solution has finished running. |
| /UseEnv | Causes the IDE to use PATH, INCLUDE, and LIB environment variables for C++ compilation, instead of the settings specified in the VC++ Directories section of Projects options in the Options dialog box. This switch is installed with the Desktop development with C++ workload. For more information, see Setting the Path and Environment Variables for Command-Line Builds . |
| /Edit | Opens the specified files in a running instance of this application. If there are no running instances, it starts a new instance with a simplified window layout. |
| /SafeMode | Starts Visual Studio in safe mode, and loads only the default environment and services, and shipped versions of third-party packages. |
| /ResetSkipPkgs | Clears all SkipLoading tags that have been added to VSPackages by users who want to avoid loading problem VSPackages. |
| /Setup | Forces Visual Studio to merge resource metadata that describes menus, toolbars, and command groups, from all VSPackages available. You must run this command as an administrator. |

The following command-line switches do not display the IDE.

| COMMAND LINE SWITCH | DESCRIPTION |
|---------------------|-------------|
| | |

| COMMAND LINE SWITCH | DESCRIPTION |
|---------------------|--|
| /? | Displays help for devenv switches in the Command Prompt window . Devenv /? |
| /Build | Builds the specified solution or project according to the configuration of the specified solution. Devenv myproj.csproj /build |
| /Clean | Deletes any files created by the build command, without affecting source files. Devenv myproj.csproj /clean |
| /Deploy | Builds the solution, along with files necessary for deployment, according to the solutions configuration. Devenv myproj.csproj /deploy |
| /Diff | Compares two files. Takes four parameters: SourceFile, TargetFile, SourceDisplayName (optional), TargetDisplayName (optional). |
| /Out | Lets you specify a file to receive errors when you build. Devenv myproj.csproj /build /out log.txt |
| /Project | The project to build, clean, or deploy. You can use this switch only if you have also supplied the /build, /rebuild, /clean, or /deploy switch. |
| /ProjectConfig | Specifies the project configuration to build or deploy. You can use this switch only if you have also supplied the /project switch. |
| /Rebuild | Cleans and then builds the specified solution or project according to the configuration of the specified solution. |
| /ResetSettings | Restores Visual Studio default settings. Optionally resets the settings to the specified .vssettings file. |
| /Upgrade | Upgrades the specified solution file and all its project files, or the specified project file, to the current Visual Studio formats for these files. |

See also

- [General, Environment, Options Dialog Box](#)

/? (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Displays a message box listing all `devenv` switches, with a brief description of each one.

Syntax

```
devenv /?
```

See Also

[Devenv Command Line Switches](#)

/Build (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Builds a solution using a specified solution configuration file.

Syntax

```
Devenv SolutionName /build SolnConfigName [/project ProjName [/projectconfig ProjConfigName]]
```

Arguments

`SolutionName`

Required. The full path and name of the solution file.

`SolnConfigName`

Required. The name of the solution configuration that will be used to build the solution named in `SolutionName`.

`/project ProjName`

Optional. The path and name of a project file within the solution. You can enter a relative path from the `SolutionName` folder to the project file, or the project's display name, or the full path and name of the project file.

`/projectconfig ProjConfigName`

Optional. The name of a project build configuration to be used when building the `/project` named.

Remarks

This switch performs the same function as the **Build Solution** menu command within the integrated development environment (IDE).

Enclose strings that include spaces in double quotation marks.

Summary information for builds, including errors, can be displayed in the **Command** window, or in any log file specified with the `/out` switch.

This command only builds projects that have changed since the last build. To build all projects in a solution, use [/Rebuild \(devenv.exe\)](#).

Example

This example builds the project `CSharpConsoleApp`, using the `Debug` project build configuration within the `Debug` solution configuration of `MySolution`.

```
devenv "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln"  
/build Debug /project "CSharpWinApp\CSharpWinApp.csproj" /projectconfig Debug
```

See Also

[Building and Cleaning Projects and Solutions in Visual Studio](#)

[Devenv Command Line Switches](#)

[/Rebuild \(devenv.exe\)](#)

/Clean (devenv.exe)

/Out (devenv.exe)

/Clean (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Cleans all intermediary files and output directories.

Syntax

```
devenv FileName /Clean [ /project projectnameorfile [/projectconfig name] ]
```

Arguments

`FileName`

Required. The full path and name of the solution file or project file.

`/project ProjName`

Optional. The path and name of a project file within the solution. You can enter a relative path from the `SolutionName` folder to the project file, or the project's display name, or the full path and name of the project file.

`/projectconfig ProjConfigName`

Optional. The name of a project build configuration to be used when cleaning the `/project` named.

Remarks

This switch performs the same function as the **Clean Solution** menu command within the integrated development environment (IDE).

Enclose strings that include spaces in double quotation marks.

Summary information for cleans and builds, including errors, can be displayed in the **Command** window, or in any log file specified with the `/out` switch.

Example

The first example cleans the `Mysolution` solution, using the default configuration specified in the solution file.

The second example cleans the project `CSharpConsoleApp`, using the `Debug` project build configuration within the `Debug` solution configuration of `MySolution`.

```
Devenv "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln"  
/Clean  
  
devenv "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln"  
/Clean /project "CSharpWinApp\CSharpWinApp.csproj" /projectconfig "Debug"
```

See Also

[Devenv Command Line Switches](#)

[/Build \(devenv.exe\)](#)

[/Rebuild \(devenv.exe\)](#)

[/Out \(devenv.exe\)](#)

/Command (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Executes the specified command after launching the Visual Studio integrated development environment (IDE).

Syntax

```
devenv /command CommandName
```

Arguments

CommandName

Required. The complete name of a Visual Studio command or its alias, enclosed in double quotation marks. For more information about command and alias syntax, see [Visual Studio Commands](#).

Remarks

After startup is complete, the IDE executes the named command. If you use this switch, the IDE does not display the Visual Studio Start Page on startup.

If an add-in exposes a command, you can use this switch to launch the add-in from the command line. For more information, see [How to: Control Add-Ins By Using the Add-In Manager](#).

Example

This example launches Visual Studio and automatically runs the macro Open Favorite Files.

```
devenv /command "Macros.MyMacros.Module1.OpenFavoriteFiles"
```

See Also

[Devenv Command Line Switches](#)

[Visual Studio Command Aliases](#)

/DebugExe (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Opens the specified executable file to be debugged.

Syntax

```
Devenv /debugexe ExecutableFile
```

Arguments

`ExecutableFile`

Required. The path and file name of an .exe file.

If the .exe file is not found or does not exist, no warning or error is displayed and Visual Studio starts normally.

Remarks

Any strings following the `ExecutableFile` parameter are passed to that file as arguments.

Example

The following example opens the file `MyApplication.exe` for debugging.

```
Devenv.exe /debugexe MyApplication.exe
```

See Also

[Devenv Command Line Switches](#)

/Deploy (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Deploys a solution after a build or rebuild. Applies to managed code projects only.

Syntax

```
devenv SolutionName /deploy SolnConfigName [/project ProjName] [/projectconfig ProjConfigName] [/out LogFileName]
```

Arguments

`SolnConfigName`

Required. The name of the solution configuration that will be used to build the solution named in `SolutionName`.

`SolutionName`

Required. The full path and name of the solution file.

`/project ProjName`

Optional. The path and name of a project file within the solution. You can enter a relative path from the `SolutionName` folder to the project file, or the project's display name, or the full path and name of the project file.

`/projectconfig ProjConfigName`

Optional. The name of a project build configuration to be used when building the `/project` named.

Remarks

The specified project must be a deployment project. If the specified project is not a deployment project, when the project that has been built is passed to be deployed, it fails with an error.

Enclose strings that include spaces in double quotation marks.

Summary information for builds, including errors, can be displayed in the **Command** window, or in any log file specified with the `/out` switch.

Example

This example deploys the project `CSharpConsoleApp`, using the `Release` project build configuration within the `Release` solution configuration of `MySolution`.

```
devenv "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln"  
/deploy Release /project "CSharpWinApp\CSharpWinApp.csproj" /projectconfig Release
```

See Also

[Devenv Command Line Switches](#)

[/Project \(devenv.exe\)](#)

[/Build \(devenv.exe\)](#)

[/Clean \(devenv.exe\)](#)

/Rebuild (devenv.exe)

/Out (devenv.exe)

/Diff

12/22/2017 • 1 min to read • [Edit Online](#)

Compares two files. The differences are displayed in a special Visual Studio window.

Syntax

```
devenv /Diff SourceFile, TargetFile, [SourceDisplayName],[TargetDisplayName]
```

Arguments

SourceFile

Required. The full path and name of the first file to be compared.

TargetFile

Required. The full path and name of the second file to be compared

SourceDisplayName

Optional. The display name of the first file.

TargetDisplayName

Optional. The display name of the second file.

/Edit (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Opens the specified file in an existing instance of Visual Studio.

Syntax

```
Devenv /edit [file1[ file2]]
```

Arguments

`file1`

Optional. The file to open in an existing instance of Visual Studio. If no instance of Visual Studio exists, a new instance is created with a simplified window layout, and `file1` is opened in the new instance.

`file2`

Optional. One or more additional files to open in the existing instance of Visual Studio.

Remarks

If no file is specified and there is an existing instance of Visual Studio, the existing instance of Visual Studio receives focus. If no file is specified and there is no existing instance of Visual Studio, a new instance of Visual Studio is created with a simplified window layout.

If the existing instance of Visual Studio is in a modal state, for example, if the [Options dialog box](#) is open, the file will open in the existing instance when Visual Studio exits the modal state.

Example

This example opens the file `MyFile.cs` in an existing instance of Visual Studio or opens the file in a new instance of Visual Studio if one does not already exist.

```
devenv /edit MyFile.cs
```

See Also

[Devenv Command Line Switches](#)

/LCID (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Sets the default language used for text, currency, and other values within the integrated development environment (IDE).

Syntax

```
devenv {/LCID|/1} LocaleID
```

Arguments

LocaleID

Required. The LCID (locale ID) of the language you specify.

Remarks

Loads the IDE and sets the default natural language for the environment. This change is persisted between sessions and reflected on the **International Settings** pane of the **Environment** options in the **Options** dialog box in the IDE.

If the specified language is not available on the user's system, the /LCID switch is ignored.

The following table lists the LCIDs of the languages supported by Visual Studio.

| LANGUAGE | LCID |
|-----------------------|------|
| Chinese (Simplified) | 2052 |
| Chinese (Traditional) | 1028 |
| English | 1033 |
| French | 1036 |
| German | 1031 |
| Italian | 1040 |
| Japanese | 1041 |
| Korean | 1042 |
| Spanish | 3082 |

Example

This example loads the IDE with English resources strings.

```
devenv /LCID 1033
```

See Also

[Devenv Command Line Switches](#)

[International Settings, Environment, Options Dialog Box](#)

[Customizing window layouts](#)

/Log (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Logs all activity to the log file for troubleshooting. This file appears after you've called `devenv /log` at least once. By default, the log file is:

`%APPDATA%\Microsoft\VisualStudio\Version\ActivityLog.xml`

where *Version* is the Visual Studio version. However, you may specify a different path and file name.

Syntax

```
Devenv /log Path\NameOfFile
```

Remarks

This switch must appear at the end of the command line, after all other switches.

The log is written for all instances of Visual Studio that you've invoked with the `/log` switch. It doesn't log instances of Visual Studio that you've invoked without the switch.

See Also

[Devenv Command Line Switches](#)

/Out (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Specifies a file to store and display errors when you run, build, rebuild, or deploy a solution.

Syntax

```
devenv /out FileName
```

Arguments

FileName

Required. The path and name of the file to receive errors when you build an executable.

Remarks

If a file name that does not exist is specified, the file is created automatically. If the file already exists, the results are appended to the existing contents of the file.

Command line build errors are displayed in the **Command** window and the Solution Builder view of the **Output** window. This option is useful if you are running unattended builds and need to see the results.

Example

This example runs `MySolution` and writes errors to the file `MyErrorLog.txt`.

```
devenv /run "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln" /out "C:\MyErrorLog.txt"
```

See Also

[Devenv Command Line Switches](#)

[/Run \(devenv.exe\)](#)

[/Build \(devenv.exe\)](#)

[/Rebuild \(devenv.exe\)](#)

[/Deploy \(devenv.exe\)](#)

/Project (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Identifies a single project within the specified solution configuration to build, clean, rebuild, or deploy.

Syntax

```
devenv SolutionName {/build|/clean|/rebuild|/deploy} SolnConfigName  
[/project ProjName] [/projectconfig ProjConfigName]
```

Arguments

/build

Builds the project specified by `/project` `ProjName`.

/clean

Cleans all intermediary files and output directories created during a build.

/rebuild

Cleans then builds the project specified by `/project` `ProjName`.

/deploy

Specifies that the project be deployed after a build or rebuild.

`SolnConfigName`

Required. The name of the solution configuration that will be applied to the solution named in `SolutionName`.

`SolutionName`

Required. The full path and name of the solution file.

`/project` `ProjName`

Optional. The path and name of a project file within the solution. You can enter a relative path from the `SolutionName` folder to the project file, or the project's display name, or the full path and name of the project file.

`/projectconfig` `ProjConfigName`

Optional. The name of a project build configuration to be applied to the `/project` named.

Remarks

- Must be used part of a `devenv /build`, `/clean`, `/rebuild`, or `/deploy` command.
- Enclose strings that include spaces in double quotation marks.
- Summary information for builds, including errors, can be displayed in the **Command** window, or in any log file specified with the `/out` switch.

Example

This example builds the project `csharpConsoleApp`, using the `Debug` project build configuration within the `Debug` solution configuration of `MySolution`.

```
devenv "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln"  
/build Debug /project "CSharpWinApp\CSharpWinApp.csproj" /projectconfig Debug
```

See Also

[Devenv Command Line Switches](#)

[/ProjectConfig \(devenv.exe\)](#)

[/Build \(devenv.exe\)](#)

[/Clean \(devenv.exe\)](#)

[/Rebuild \(devenv.exe\)](#)

[/Deploy \(devenv.exe\)](#)

[/Out \(devenv.exe\)](#)

/ProjectConfig (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Specifies a project build configuration to be applied when you build, clean, rebuild, or deploy the project named in the `/project` argument.

Syntax

```
devenv SolutionName {/build|/clean|/rebuild|/deploy} SolnConfigName [/project ProjName] [/projectconfig ProjConfigName]
```

Arguments

`/build`

Builds the project specified by `/project` `ProjName`.

`/clean`

Cleans all intermediary files and output directories created during a build.

`/rebuild`

Cleans then builds the project specified by `/project` `ProjName`.

`/deploy`

Specifies that the project be deployed after a build or rebuild.

`SolnConfigName`

Required. The name of the solution configuration that will be applied to the solution named in `SolutionName`.

`SolutionName`

Required. The full path and name of the solution file.

`/project` `ProjName`

Optional. The path and name of a project file within the solution. You can enter a relative path from the `SolutionName` folder to the project file, or the project's display name, or the full path and name of the project file.

`/projectconfig` `ProjConfigName`

Optional. The name of a project build configuration to be applied to the `/project` named.

Remarks

- Must be used with the `/project` switch as part of a `devenv /build`, `/clean`, `/rebuild`, or `/deploy` command.
- Enclose strings that include spaces in double quotation marks.
- Summary information for builds, including errors, can be displayed in the **Command** window, or in any log file specified with the `/out` switch.

Example

This example builds the project `csharpConsoleApp`, using the `Debug` project build configuration within the `Debug`

solution configuration of `MySolution`.

```
devenv "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln"  
/build Debug /project "CSharpWinApp\CSharpWinApp.csproj" /projectconfig Debug
```

See Also

[Devenv Command Line Switches](#)

[/Project \(devenv.exe\)](#)

[/Build \(devenv.exe\)](#)

[/Clean \(devenv.exe\)](#)

[/Rebuild \(devenv.exe\)](#)

[/Deploy \(devenv.exe\)](#)

[/Out \(devenv.exe\)](#)

/Rebuild (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Cleans and then builds the specified solution configuration.

Syntax

```
devenv SolutionName /rebuild SolnConfigName [/project ProjName] [/projectconfig ProjConfigName]
```

Arguments

`SolnConfigName`

Required. The name of the solution configuration that will be used to rebuild the solution named in `SolutionName`.

`SolutionName`

Required. The full path and name of the solution file.

`/project ProjName`

Optional. The path and name of a project file within the solution. You can enter a relative path from the `SolutionName` folder to the project file, or the project's display name, or the full path and name of the project file.

`/projectconfig ProjConfigName`

Optional. The name of a project build configuration to be used when rebuilding the `/project` named.

Remarks

- This switch performs the same function as the **Rebuild Solution** menu command within the integrated development environment (IDE).
- Enclose strings that include spaces in double quotation marks.
- Summary information for cleans and builds, including errors, can be displayed in the **Command** window, or in any log file specified with the `/out` switch.

Example

This example cleans and rebuilds the project `CSharpWinApp`, using the `Debug` project build configuration within the `Debug` solution configuration of `MySolution`.

```
devenv "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln"  
/rebuild Debug /project "CSharpWinApp\CSharpWinApp.csproj" /projectconfig Debug
```

See Also

[Devenv Command Line Switches](#)

[/Build \(devenv.exe\)](#)

[/Clean \(devenv.exe\)](#)

[/Out \(devenv.exe\)](#)

/ResetSettings (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Restores Visual Studio default settings and automatically launches the Visual Studio IDE. Optionally resets the settings to a specified .vssettings file.

The default settings are determined by the profile that was selected when Visual Studio was first launched.

Syntax

```
Devenv /ResetSettings SettingsFile
```

Arguments

`SettingsFile`

The full path and name of the .vssettings file to apply to Visual Studio.

To restore the General Development Settings profile, use `General`.

Remarks

If no `SettingsFile` is specified, you will be prompted to select a default collection of settings the next time you start Visual Studio.

Example

The following command line applies the settings stored in the file `MySettings.vssettings`.

```
Devenv.exe /ResetSettings "C:\My Files\MySettings.vssettings"
```

See Also

[Personalize the Visual Studio IDE](#)

[Devenv Command Line Switches](#)

/ResetSkipPkgs (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Clears all options to skip loading added to VSPackages by users wishing to avoid loading problem VSPackages, then starts Visual Studio.

Syntax

```
Devenv /ResetSkipPkgs
```

Remarks

The presence of a SkipLoading tag disables the loading of a VSPackage; clearing the tag re-enables the loading of the VSPackage.

Example

The following example clears all SkipLoading tags.

```
Devenv.exe /ResetSkipPkgs
```

See Also

[Devenv Command Line Switches](#)

/Run (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Compiles and runs the specified project or solution.

Syntax

```
devenv {/run|/r} {SolutionName|ProjectName}
```

Arguments

`SolutionName`

Required. The full path and name of a solution file.

`ProjectName`

Required. The full path and name of a project file.

Remarks

Compiles and runs the specified project or solution according to the settings specified for the active solution configuration. This switch launches the integrated development environment (IDE) and leaves it active after the project or solution has completed running.

- Enclose strings that include spaces in double quotation marks.
- Summary information, including errors, can be displayed in the **Command** window, or in any log file specified with the `/out` switch.

Example

This example runs the solution `MySolution` using the active deployment configuration.

```
devenv /run "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln"
```

See Also

[Devenv Command Line Switches](#)

[/Runexit \(devenv.exe\)](#)

[/Build \(devenv.exe\)](#)

[/Rebuild \(devenv.exe\)](#)

[/Out \(devenv.exe\)](#)

/Runexit (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Compiles and runs the specified project or solution, and then closes the integrated development environment (IDE).

Syntax

```
devenv /runexit {SolutionName|ProjectName}
```

Arguments

`SolutionName`

Required. The full path and name of a solution file.

`ProjectName`

Required. The full path and name of a project file.

Remarks

Compiles and runs the specified project or solution according to the settings specified for the active solution configuration. This switch minimizes the IDE while the project or solution is run, and it closes the IDE after the project or solution has completed running.

- Enclose strings that include spaces in double quotation marks.
- Summary information, including errors, can be displayed in the **Command** window, or in any log file specified with the `/out` switch.

Example

This example runs the solution `MySolution` in a minimized IDE using the active deployment configuration, and then closes the IDE.

```
devenv /runexit "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln"
```

See Also

[Devenv Command Line Switches](#)

[/Run \(devenv.exe\)](#)

[/Build \(devenv.exe\)](#)

[/Rebuild \(devenv.exe\)](#)

[/Out \(devenv.exe\)](#)

/SafeMode (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Starts Visual Studio in safe mode, loading only the default environment and services.

Syntax

```
devenv /SafeMode
```

Remarks

This switch prevents all third-party VSPackages from loading when Visual Studio starts, thus ensuring stable execution.

Description

The following example starts Visual Studio in safe mode.

Code

```
Devenv.exe /SafeMode
```

See Also

[Devenv Command Line Switches](#)

/Setup (devenv.exe)

3/2/2018 • 1 min to read • [Edit Online](#)

The /Setup switch causes Visual Studio to merge the resource metadata that describes menus, toolbars, and command groups, from all available VSPackages.

Syntax

```
devenv /setup
```

Remarks

This switch takes no arguments. The `devenv /setup` command is typically given as the last step of the installation process. Use of the `/setup` switch does not start Visual Studio.

NOTE

You must run `devenv` as an administrator in order to use the `/setup` switch.

Example

This example shows the last step in the installation of a version of Visual Studio that includes VSPackages.

```
devenv /setup
```

See also

[Devenv Command Line Switches](#)

/Upgrade (devenv.exe)

12/22/2017 • 1 min to read • [Edit Online](#)

Updates the solution file and all of its project files, or the project file specified, to the current Visual Studio formats for these files.

Syntax

```
devenv SolutionFile | ProjectFile /upgrade
```

Arguments

`SolutionFile`

Required if you are upgrading an entire solution and its projects. The path and name of a solution file. You can enter just the name of the solution file, or a full path and the name of the solution file. If the folder or file named does not yet exist, it will be created.

`ProjectFile`

Required if you are upgrading a single project. The path and name of a project file within the solution. You can enter just the name of the project file, or a full path and the name of the project file. If the folder or file named does not yet exist, it will be created.

Remarks

Backups are automatically created and copied to a directory named Backup that is created in the current directory.

Source-controlled solutions or projects must be checked out before they can be upgraded.

Using the `/upgrade` switch does not start Visual Studio. Results of the upgrade can be seen in the Upgrade Report for the development language of the solution or project. No error or usage info is returned. For more information on upgrading projects in Visual Studio, see [Port, Migrate, and Upgrade Visual Studio Projects](#).

Example

This example upgrades a solution file named "MyProject.sln" in your default folder for Visual Studio solutions.

```
devenv "MyProject.sln" /upgrade
```

See Also

[Devenv Command Line Switches](#)

/UseEnv (devenv.exe)

3/2/2018 • 1 min to read • [Edit Online](#)

Starts Visual Studio and loads environmental variables into the **VC++ Directories** dialog box.

NOTE

This switch is installed with the **Desktop development with C++** workload.

Syntax

```
Devenv /useenv
```

Example

The following example starts Visual Studio and loads environment variables into the **VC++ Directories** dialog box.

```
Devenv.exe /useenv
```

See also

- [Devenv Command Line Switches](#)

Visual Studio Commands

12/22/2017 • 3 min to read • [Edit Online](#)

Visual Studio commands allow you to invoke a command from the **Command** window, **Immediate** window, or **Find/Command** box. In each case, the greater than sign (>) is used to indicate that a command rather than a search or debug operation is to follow.

You can find a complete list of commands and their syntax in the **Keyboard, Environment Options** dialog box.

The escape character for Visual Studio commands is a caret (^) character, which means that the character immediately following it is interpreted literally, rather than as a control character. This can be used to embed straight quotation marks ("), spaces, leading slashes, carets, or any other literal characters in a parameter or switch value, with the exception of switch names. For example,

```
>Edit.Find ^^t /regex
```

A caret functions the same whether it is inside or outside quotation marks. If a caret is the last character on the line, it is ignored.

In localized versions of the IDE, command names can be entered either in the native language of the IDE or in English. For example, you can type either `File.NewFile` or `Fichier.NouveauFichier` in the French IDE to execute the same command.

Many commands have aliases. For a list of command aliases, see [Visual Studio Command Aliases](#).

The following commands take arguments and/or switches.

| COMMAND NAME | DESCRIPTION |
|----------------------|--|
| Add Existing Item | Adds an existing file to the current solution and opens it. |
| Add Existing Project | Adds an existing project to the current solution. |
| Add New Item | Adds a new solution item, such as an .htm, .css, .txt, or frameset to the current solution and opens it. |
| Alias | Creates a new alias for a complete command, complete command and arguments, or even another alias. |
| Evaluate Statement | Evaluates and displays the given statement. |
| Find | Searches files using a subset of the options available on the Find and Replace control. |
| Find in Files | Searches files using a subset of the options available on the Find in Files . |
| Go To | Moves the cursor to the specified line. |
| List Call Stack | Displays the current call stack. |

| COMMAND NAME | DESCRIPTION |
|---|---|
| List Disassembly | Begins the debug process and allows you to specify how errors are handled. |
| List Memory | Displays the contents of the specified range of memory. |
| List Modules | Lists the modules for the current process. |
| List Registers | Displays a list of registers. |
| List Source | Displays the specified lines of source code. |
| List Threads | Displays a list of the threads in the current program. |
| Log Command Window Output | Copies all input and output from the Command window into a file. |
| New File | Creates a new file and adds it to the currently selected project. |
| Open File | Opens an existing file and allows you to specify an editor. |
| Open Project | Opens an existing project and allows you to add the project to the current solution. |
| Open Solution | Opens an existing solution. |
| Print | Evaluates the expression and displays the results or the specified text. |
| Quick Watch Command | Displays the selected or specified text in the Expression field of the Quick Watch dialog box. |
| Replace | Replaces text in files using a subset of the options available on the Find and Replace control. |
| Replace in Files | Replaces text in files using a subset of the options available in the Replace in Files . |
| Set Current Stack Frame | Allows you to view a particular stack frame. |
| Set Current Thread | Allows you to view a particular thread. |
| Set Radix | Determines the number of bytes to view. |
| Shell | Launches programs from within Visual Studio as though the command has been executed from the command prompt. |
| ShowWebBrowser Command | Displays the URL you specify in a Web browser window either within the integrated development environment (IDE) or external to the IDE. |

| COMMAND NAME | DESCRIPTION |
|-----------------------------------|---|
| Start | Begins the debug process and allows you to specify how errors are handled. |
| Path | Sets the list of directories for the debugger to search for symbols. |
| Toggle Breakpoint | Turns the breakpoint either on or off, depending on its current state, at the current location in the file. |
| Watch Command | Creates and opens a specified instance of a Watch window. |

See Also

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Visual Studio Command Aliases

12/22/2017 • 3 min to read • [Edit Online](#)

Aliases provide a means for entering a command into the **Find/Command** box or **Command** window by shortening the text needed to execute the command. For example, instead of entering `>File.OpenFile` to display the **Open File** dialog box, you can use the pre-defined alias `>of`.

Type `alias` in the **Command** window to display a list of the current aliases and their definitions. Type `>cls` to clear the contents of the **Command** window. If you want to see an alias for a specific command, type `alias <command name>`.

You can easily create your own alias for one of the Visual Studio commands (with or without arguments). For example, the syntax for aliasing `File.NewFile MyFile.txt` is `alias MyAlias File.NewFile MyFile.txt`. You can delete one of your aliases with `alias <alias name> /delete`.

The table below contains a list of the pre-defined Visual Studio command aliases. Some command names have more than one pre-defined alias. Click the links for the command names below to display detailed topics that explain the correct syntax, arguments, and switches for those commands.

| COMMAND NAME | ALIAS | COMPLETE NAME |
|---------------------|-----------|--------------------------|
| Print Command | ? | Debug.Print |
| Quick Watch Command | ?? | Debug.Quickwatch |
| Add New Project | AddProj | File.AddNewProject |
| Alias Command | Alias | Tools.Alias |
| Autos window | Autos | Debug.Autos |
| Breakpoints window | bl | Debug.Breakpoints |
| Toggle Breakpoint | bp | Debug.ToggleBreakPoint |
| Call Stack window | CallStack | Debug.CallStack |
| Clear Bookmarks | ClearBook | Edit.ClearBookmarks |
| Close | Close | File.Close |
| Close All Documents | CloseAll | Window.CloseAllDocuments |
| Clear All | cls | Edit.ClearAll |
| Command mode | cmd | View.CommandWindow |
| View Code | code | View.ViewCode |
| List Memory Command | d | Debug.ListMemory |

| COMMAND NAME | ALIAS | COMPLETE NAME |
|---|------------|--|
| List Memory Command as ANSI | da | Debug.ListMemory /Ansi |
| List Memory Command One Byte format | db | Debug.ListMemory /Format:OneByte |
| List Memory Command as ANSI with Four Byte format | dc | Debug.ListMemory /Format:FourBytes /Ansi |
| List Memory Command Four Byte format | dd | Debug.ListMemory /Format:FourBytes |
| Delete to BOL | DelBOL | Edit.DeleteToBOL |
| Delete to EOL | DelEOL | Edit.DeleteToEOL |
| Delete Horizontal Whitespace | DelHSp | Edit.DeleteHorizontalWhitespace |
| View Designer | designer | View.ViewDesigner |
| List Memory Command Float format | df | Debug.ListMemory/Format:Float |
| Disassembly window | disasm | Debug.Disassembly |
| List Memory Command Eight Byte format | dq | Debug.ListMemory /Format:EightBytes |
| List Memory Command as Unicode | du | Debug.ListMemory /Unicode |
| Evaluate Statement Command | eval | Debug.EvaluateStatement |
| Exit | Exit | File.Exit |
| Format Selection | format | Edit.FormatSelection |
| Full Screen | FullScreen | View.FullScreen |
| Start Command | g | Debug.Start |
| Go To Command | GotoLn | Edit.GoTo |
| Go to Brace | GotoBrace | Edit.GotoBrace |
| F1Help | Help | Help.F1Help |
| Immediate Mode | immed | Tools.ImmediateMode |
| Insert File as Text | InsertFile | Edit.InsertFileAsText |
| List Call Stack Command | kb | Debug.ListCallStack |
| Make Lower Case | Lcase | Edit.MakeLowercase |

| COMMAND NAME | ALIAS | COMPLETE NAME |
|---|---------------------------|------------------------------|
| Cut Line | LineCut | Edit.LineCut |
| Delete Line | LineDel | Edit.LineDelete |
| List Members | ListMembers | Edit.ListMembers |
| Locals window | Locals | Debug.Locals |
| Log Command Window Output Command | Log | Tools.LogCommandWindowOutput |
| Command Window Mark Mode | mark | Tools.CommandWindowMarkMode |
| Memory window | Memory Memory1 | Debug.Memory1 |
| Memory Window 2 | Memory2 | Debug.Memory2 |
| Memory Window 3 | Memory3 | Debug.Memory3 |
| Memory Window 4 | Memory4 | Debug.Memory4 |
| Set Radix Command | n | Debug.SetRadix |
| ShowWebBrowser Command | nav navigate | View.ShowWebBrowser |
| Next Bookmark | NextBook | Edit.NextBookmark |
| New File Command | nf | File.NewFile |
| New Project | np NewProj | File.NewProject |
| Open File Command | of Open | File.OpenFile |
| Open Project Command | op | File.OpenProject |
| Collapse to Definitions/Stop Outlining | OutlineDefs StopOutlining | Edit.CollapsetoDefinitions |
| Step Over | p | Debug.StepOver |
| Parameter Information | ParamInfo | Edit.ParameterInfo |
| Step Out | pr | Debug.StepOut |
| Previous Bookmark | PrevBook | Edit.PreviousBookmark |
| Print File | print | File.Print |
| Properties Window | props | View.PropertiesWindow |
| Stop | q | Debug.StopDebugging |

| COMMAND NAME | ALIAS | COMPLETE NAME |
|--|------------|---|
| Redo | redo | Edit.Redo |
| Registers window | registers | Debug.Registers |
| Run to Cursor | rtc | Debug.RunToCursor |
| Save Selected Items | save | File.SaveSelectedItems |
| Save All | SaveAll | File.WriteAll |
| Save As | SaveAs | File.SaveSelectedItemsAs |
| Shell Command | shell | Tools.Shell |
| Stop Find In Files | StopFind | Edit.FindInFiles /stop |
| Swap Anchor | SwapAnchor | Edit.SwapAnchor |
| Step Into | t | Debug.StepInto |
| Tabify Selection | tabify | Edit.TabifySelection |
| Tasklist window | TaskList | View.TaskList |
| Threads window | Threads | Debug.Threads |
| Tile Horizontally | TileH | Window.TileHorizontally |
| Tile Vertically | TileV | Window.TileVertically |
| Toggle Bookmark | ToggleBook | Edit.ToggleBookmark |
| Toolbox window | toolbox | View.Toolbox |
| List Disassembly Command | u | Debug.ListDisassembly |
| Make Uppercase | Ucase | Edit.MakeUppercase |
| Undo | undo | Edit.Undo |
| Untabify Selection | Untabify | Edit.UntabifySelection |
| Watch window | Watch | Debug.WatchN |
| Toggle Word Wrap | WordWrap | Edit.ToggleWordWrap |
| List Processes | | Debug.ListProcesses |
| List Threads Command | ~ ~*k ~*kb | Debug.ListThreads Debug.ListThreads /AllThreads |

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

Add Existing Item Command

12/22/2017 • 1 min to read • [Edit Online](#)

Adds an existing file to the current solution and opens it.

Syntax

```
File.AddExistingItem filename [/e:editorname]
```

Arguments

`filename`

Required. The full path and file name, with extension, of the item to add to the current solution. If the file path or file name contains spaces, enclose the entire path in quotation marks.

Switches

`/e: editorname`

Optional. Name of the editor in which the file will be opened. If the argument is specified but no editor name is supplied, the **Open With** dialog box appears.

The `/e: editorname` argument syntax uses the editor names as they appear in the **Open With Dialog Box**, enclosed in quotation marks. For example, to open a style sheet in the source code editor, you would enter the following for the `/e: editorname` argument.

```
/e:"Source Code (text) Editor"
```

Remarks

Autocompletion tries to locate the correct path and file name as you type.

Example

This example adds the file, Form1.frm, to the current solution.

```
>File.AddExistingItem "C:\public\solution files\Form1.frm"
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Add Existing Project Command

12/22/2017 • 1 min to read • [Edit Online](#)

Adds an existing project to the current solution.

Syntax

```
File.AddExistingProject filename
```

Arguments

`filename`

Optional. The full path and project name, with extension, of the project to add to the solution.

If the `filename` argument includes spaces, it must be enclosed in quotation marks.

If no filename is specified, the command will open the file dialog so that user can pick a project.

Remarks

Auto completion tries to locate the correct path and file name as you type.

Example

This example adds the Visual Basic project, TestProject1, to the current solution.

```
>File.AddExistingProject "c:\visual studio projects\TestProject1.vbproj"
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Add New Item Command

12/22/2017 • 1 min to read • [Edit Online](#)

Adds a new solution item, such as an .htm, .css, .txt, or frameset to the current solution and opens it.

Syntax

```
File.AddNewItem [filename] [/t:templatename] [/e:editorname]
```

Arguments

`filename`

Optional. The path and file name of the item to add to the solution.

Switches

`/t: templatename`

Optional. Specifies the type of file to be created. If no template name is given, a text file is created by default.

The `/t: templatename` argument syntax mirrors the information found in the **Add New Solution Item** dialog box.

You must enter the complete category followed by the file type, separating the category name from the file type by a backslash (`\`) and enclosing the entire string in quotation marks.

For example, to create a new text file, you would enter the following for the `/t: templatename` argument.

```
/t:"General\Style Sheet"
```

`/e: editorname`

Optional. The name of the editor in which the file will be opened. If the argument is specified but no editor name is supplied, the **Open With** dialog box appears.

The `/e: editorname` argument syntax uses the editor names as they appear in the **Open With Dialog Box**, enclosed in quotation marks.

For example, to open a style sheet in the source code editor, you would enter the following for the `/e: editorname` argument.

```
/e:"Source Code (text) Editor"
```

Example

This example adds a new solution item, MyHTMLpg, to the current solution.

```
>File.AddNewItem MyHTMLpg /t:"General\HTML Page"
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Alias Command

12/22/2017 • 1 min to read • [Edit Online](#)

Creates a new alias for a complete command, complete command and arguments, or another alias.

TIP

Typing `>alias` without any arguments displays the current list of aliases and their definitions.

Syntax

```
Tools.Alias [/delete] [/reset] [aliasname] [aliasstring]
```

Arguments

`aliasname`

Optional. The name for the new alias. If no value is supplied for `aliasname`, a list of the current aliases and their definitions appears.

`aliasstring`

Optional. The complete command name or existing alias and any parameters that you want to create as an alias. If no value is supplied for `aliasstring`, the alias name and alias string for the specified alias displays.

Switches

`/delete` or `/del` or `/d`

Optional. Deletes the specified alias, removing it from autocompletion.

`/reset`

Optional. Resets the list of pre-defined aliases to its original settings. That is, it restores all pre-defined aliases and removes all user-defined aliases.

Remarks

Since aliases represent commands, they must be located at the beginning of the command line.

When issuing this command, you should include the switches immediately after the command, not after the aliases, otherwise the switch itself will be included as part of the alias string.

The `/reset` switch asks for a confirmation before the aliases are restored. There is no short form of `/reset`.

Examples

This example creates a new alias, `upper`, for the complete command `Edit.MakeUpperCase`.

```
>Tools.Alias upper Edit.MakeUpperCase
```

This example deletes the alias, `upper`.

```
>Tools.alias /delete upper
```

This example displays a list of all current aliases and definitions.

```
>Tools.Alias
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Evaluate Statement Command

12/22/2017 • 1 min to read • [Edit Online](#)

Evaluates and displays the given statement.

Syntax

```
Debug.EvaluateStatement text
```

Arguments

`text`

Required. The statement to evaluate.

Remarks

The window used to enter the **EvaluateStatement** command determines whether an equals sign (=) is interpreted as a comparison operator or as an assignment operator.

In the **Command** window, an equals sign (=) is interpreted as a comparison operator. So, for example, if the values of variables `a` and `b` are different, then the command

```
>Debug.EvaluateStatement(a=b)
```

will return a value of `false`.

In the **Immediate** window, by contrast, an equals sign (=) is interpreted as an assignment operator. So, for example, the command

```
>Debug.EvaluateStatement(a=b)
```

will assign to variable `a` the value of variable `b`.

Example

```
>Debug.EvaluateStatement(a+b)
```

See Also

[Print Command](#)

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Find Command

12/22/2017 • 1 min to read • [Edit Online](#)

Searches files using a subset of the options available on the **Find in Files** tab of the **Find and Replace** window.

Syntax

```
Edit.Find findwhat [/case] [/doc | /proc | /open | /sel]  
[/markall] [/options] [/reset] [/up] [/wild | /regex] [/word]
```

Arguments

findwhat

Required. The text to match.

Switches

/case or /c

Optional. Matches occur only if the uppercase and lowercase characters exactly match those specified in the **findwhat** argument.

/doc or /d

Optional. Searches the current document only. Specify only one of the available search scopes, **/doc**, **/proc**, **/open**, or **/sel**.

/markall or /m

Optional. Places a graphic on each line that contains a search match within the current document.

/open or /o

Optional. Searches all open documents as if they were one document. Specify only one of the available search scopes, **/doc**, **/proc**, **/open**, or **/sel**.

/options or /t

Optional. Displays a list of the current find option settings and does not perform a search.

/proc or /p

Optional. Searches the current procedure only. Specify only one of the available search scopes, **/doc**, **/proc**, **/open**, or **/sel**.

/reset or /e

Optional. Returns the find options to their default settings and does not perform a search.

/sel or /s

Optional. Searches the current selection only. Specify only one of the available search scopes, **/doc**, **/proc**, **/open**, or **/sel**.

/up or /u

Optional. Searches from the current location in the file toward the beginning of the file. By default, searches begin at the current location in the file and searches towards the end of the file.

/regex or /r

Optional. Uses pre-defined special characters in the `findwhat` argument as notations that represent patterns of text rather than the literal characters. For a complete list of regular expression characters, see [Regular Expressions](#).

/wild or /l

Optional. Uses pre-defined special characters in the `findwhat` argument as notations to represent a character or sequence of characters.

/word or /w

Optional. Searches only for whole words.

Example

This example performs a case-sensitive search for the word "somestring" in the currently selected section of code.

```
>Edit.Find somestring /sel /case
```

See Also

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Commands](#)

[Visual Studio Command Aliases](#)

Find in Files Command

12/22/2017 • 1 min to read • [Edit Online](#)

Search files using a subset of the options available on the **Find in Files** tab of the **Find and Replace** window.

Syntax

```
Edit.FindinFiles findwhat [/case] [/ext:extensions]
[/lookin:searchpath] [/names] [/options] [/reset] [/stop] [/sub]
[/text2] [/wild|/regex] [/word]
```

Arguments

findwhat

Required. The text to match.

Switches

/case or /c

Optional. Matches occur only if the uppercase and lowercase characters exactly match those specified in the **findwhat** argument.

/ext: extensions

Optional. Specifies the file extensions for the files to be searched. If not specified, the previous extension is used if one was previously entered.

/lookin: searchpath

Optional. Directory to search. If the path contains spaces, enclose the entire path in quotation marks.

/names or /n

Optional. Displays a list of file names that contain matches.

/options or /t

Optional. Displays a list of the current find option settings and does not perform a search.

/regex or /r

Optional. Uses pre-defined special characters in the **findwhat** argument as notations that represent patterns of text rather than the literal characters. For a complete list of regular expression characters, see [Regular Expressions](#).

/reset or /e

Optional. Returns the find options to their default settings and does not perform a search.

/stop

Optional. Halts the current search operation if one is in progress. Search ignores all other arguments when **/stop** has been specified. For example, to stop the current search you would enter the following:

```
>Edit.FindinFiles /stop
```

/sub or /s

Optional. Searches the subfolders within the directory specified in the **/lookin: searchpath** argument.

/text2 or /2

Optional. Displays the results of the search in the Find Results 2 window.

/wild or /l

Optional. Uses pre-defined special characters in the `findwhat` argument as notations to represent a character or sequence of characters.

/word or /w

Optional. Searches only for whole words.

Example

This example searches for btnCancel in all .cls files located in the folder "My Visual Studio Projects" and displays the match information in the Find Results 2 Window.

```
>Edit.FindinFiles btnCancel /lookin:"c:/My Visual Studio Projects" /ext:*.cls /text2
```

See Also

[Find in Files](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Commands](#)

[Visual Studio Command Aliases](#)

Go To Command

12/22/2017 • 1 min to read • [Edit Online](#)

Moves the cursor to the specified line.

Syntax

```
Edit.GoTo [linenumber]
```

Arguments

linenumber

Optional. An integer representing the number of the line to go to.

Remarks

The line numbering begins at one. If the value of `linenumber` is less than one, the first line displays. If the value of `linenumber` is greater than the number of the last line, the last line displays.

If a value for `linenumber` is not specified, the **Go To Line** dialog box displays.

The alias for this command is GoToLn.

Example

```
>Edit.GoTo 125
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Import and Export Settings Command

1/26/2018 • 1 min to read • [Edit Online](#)

Imports, exports, or resets Visual Studio settings.

Syntax

```
Tools.ImportandExportSettings [/export:filename | /import:filename | /reset]
```

Switches

/export: `filename`

Optional. Exports the current settings to the specified file.

/import: `filename`

Optional. Imports the settings in the specified file.

/reset

Optional. Resets the current settings.

Remarks

Running this command with no switches opens the **Import and Export Settings** wizard. For more information, see [Synchronize your settings](#).

Example

The following command exports the current settings to the file `MyFile.vssettings`.

```
Tools.ImportandExportSettings /export:"c:\Files\MyFile.vssettings"
```

See also

[Personalize the Visual Studio IDE](#)

[Visual Studio Commands](#)

List Call Stack Command

12/22/2017 • 1 min to read • [Edit Online](#)

Displays the current call stack.

Syntax

```
Debug.ListCallStack [/Count:number] [/ShowTypes:yes|no]
[/ShowNames:yes|no] [/ShowValues:yes|no] [/ShowModule:yes|no]
[/ShowLineOffset:yes|no] [/ShowByteOffset:yes|no]
[/ShowLanguage:yes|no] [/IncludeCallsAcrossThreads:yes|no]
[/ShowExternalCode:yes|no] [Thread:n] [index]
```

Arguments

`index`

Optional. Sets the current stack frame and displays no output.

Switches

Each switch can be invoked using either its complete form or a short form.

`/Count: number` [or] `/C: number`

Optional. Maximum number of call stacks to display. The default value is unlimited.

`/ShowTypes: yes | no` [or] `/T: yes | no`

Optional. Specifies whether to display parameter types. Default value is `yes`.

`/ShowNames: yes | no` [or] `/N: yes | no`

Optional. Specifies whether to display parameter names. Default value is `yes`.

`/ShowValues: yes | no` [or] `/V: yes | no`

Optional. Specifies whether to display parameter values. Default value is `yes`.

`/ShowModule: yes | no` [or] `/M: yes | no`

Optional. Specifies whether to display the module name. Default value is `yes`.

`/ShowLineOffset: yes | no` [or] `/#: yes | no`

Optional. Specifies whether to display the line offset. Default value is `no`.

`/ShowByteOffset: yes | no` [or] `/B: yes | no`

Optional. Specifies whether to display the byte offset. Default value is `no`.

`/ShowLanguage: yes | no` [or] `/L: yes | no`

Optional. Specifies whether to display the language. Default value is `no`.

`/IncludeCallsAcrossThreads: yes | no` [or] `/I: yes | no`

Optional. Specifies whether to include calls to or from other threads. Default value is `no`.

`/ShowExternalCode: yes | no`

Optional. Specifies whether to display Just My Code for the callstack. When Just My Code is off, all non-user code is displayed. When Just My Code is on, non-user code is displayed as `[external]` in the callstack output.

Thread:

Optional. Displays the callstack for thread . If no thread is specified, displays the callstack for the current thread.

Remarks

Changes made to the arguments or switches apply to future invocations of this command. If you issue `Debug.ListCallStack` by itself, the entire call stack displays. If you specify an index, for example,

```
Debug.ListCallStack 2
```

then the current stack frame is set to that frame (in this case, the second frame).

You can also write this command using its pre-defined alias, `kb`. For example, you can enter

```
kb 2
```

to set the current stack frame to the second frame.

Example

```
>Debug.CallStack /Count:4 /ShowTypes:yes
```

See Also

[List Disassembly Command](#)

[List Threads Command](#)

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

List Disassembly Command

12/22/2017 • 1 min to read • [Edit Online](#)

Begins the debug process and allows you to specify how errors are handled.

Syntax

```
Debug.ListDisassembly [/count:number] [/endaddress:expression]
[/codebytes:yes|no] [/source:yes|no] [/symbolnames:yes|no]
[/linenumbers:yes|no]
```

Switches

Each switch can be invoked using either its complete form or a short form.

/count: `number` [or] /c: `number` [or] /length: `number` [or] /l: `number`

Optional. Number of instructions to display. Default value is 8.

/endaddress: `expression` [or] /e: `expression`

Optional. Address at which to stop disassembly.

/codebytes: `yes` | `no` [or] /bytes: `yes` | `no` [or] /b: `yes` | `no`

Optional. Indicates whether to display code bytes. Default value is `no`.

/source: `yes` | `no` [or] /s: `yes` | `no`

Optional. Indicates whether to display source code. Default value is `no`.

/symbolnames: `yes` | `no` [or] /names: `yes` | `no` [or] /n: `yes` | `no`

Optional. Indicates whether to display symbols names. Default value is `yes`.

[/linenumbers: `yes` | `no`]

Optional. Enables the viewing of line numbers associated with the source code. The /source switch must have a value of `yes` to use the /linenumbers switch.

Example

```
>Debug.ListDisassembly
```

See Also

[List Call Stack Command](#)

[List Threads Command](#)

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

List Memory Command

12/22/2017 • 1 min to read • [Edit Online](#)

Displays the contents of the specified range of memory.

Syntax

```
Debug.ListMemory [/ANSI|Unicode] [/Count:number] [/Format:formattype]
[/Hex|Signed|Unsigned] [expression]
```

Arguments

`expression`

Optional. The memory address from which to begin displaying memory.

Switches

`/ANSI|Unicode`

Optional. Display the memory as characters corresponding to the bytes of memory, either ANSI or Unicode.

`/Count: number`

Optional. Determines how many bytes of memory to display, starting at `expression`.

`/Format: formattype`

Optional. Format type for viewing memory information in the **Memory** window; may be OneByte, TwoBytes, FourBytes, EightBytes, Float (32-bit), or Double (64-bit). If OneByte is used, `/Unicode` is unavailable.

`/Hex|Signed|Unsigned`

Optional. Specifies the format for viewing numbers: as signed, unsigned, or hexadecimal.

Remarks

Instead of writing out a complete **Debug.ListMemory** command with all switches, you can invoke the command using predefined aliases with certain switches preset to specified values. For example, instead of entering:

```
>Debug.ListMemory /Format:float /Count:30 /Unicode
```

you can write:

```
>df /Count:30 /Unicode
```

Here is a list of the available aliases for the **Debug.ListMemory** command:

| ALIAS | COMMAND AND SWITCHES |
|-----------------|-------------------------------------|
| <code>d</code> | <code>Debug.ListMemory</code> |
| <code>da</code> | <code>Debug.ListMemory /AnsI</code> |

| ALIAS | COMMAND AND SWITCHES |
|-----------|--|
| db | Debug.ListMemory /Format:OneByte |
| dc | Debug.ListMemory /Format:FourBytes /Ansi |
| dd | Debug.ListMemory /Format:FourBytes |
| df | Debug.ListMemory /Format:Float |
| dq | Debug.ListMemory /Format:EightBytes |
| du | Debug.ListMemory /Unicode |

Example

```
>Debug.ListMemory /Format:float /Count:30 /Unicode
```

See Also

[List Call Stack Command](#)

[List Threads Command](#)

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

List Modules Command

12/22/2017 • 1 min to read • [Edit Online](#)

Lists the modules for the current process.

Syntax

```
Debug.ListModules [/Address:yes|no] [/Name:yes|no] [/Order:yes|no]
[/Path:yes|no] [/Process:yes|no] [/SymbolFile:yes|no]
[/SymbolStatus:yes|no] [/Timestamp:yes|no] [/Version:yes|no]
```

Parameters

/Address: `yes|no`

Optional. Specifies whether to show the memory addresses of the modules. Default value is `yes`.

/Name: `yes|no`

Optional. Specifies whether to show the names of the modules. Default value is `yes`.

/Order: `yes|no`

Optional. Specifies whether to show the order of the modules. Default value is `no`.

/Path: `yes|no`

Optional. Specifies whether to show the paths of the modules. Default value is `yes`.

/Process: `yes|no`

Optional. Specifies whether to show the processes of the modules. Default value is `no`.

/SymbolFile: `yes|no`

Optional. Specifies whether to show the symbol files of the modules. Default value is `no`.

/SymbolStatus: `yes|no`

Optional. Specifies whether to show the symbol statuses of the modules. Default value is `yes`.

/Timestamp: `yes|no`

Optional. Specifies whether to show the timestamps of the modules. Default value is `no`.

/Version: `yes|no`

Optional. Specifies whether to show the versions of the modules. Default value is `no`.

Remarks

Example

This example lists the module names, addresses, and timestamps for the current process.

```
Debug.ListModules /Address:yes /Name:yes /Order:no /Path:no /Process:no /SymbolFile:no /SymbolStatus:no
/Timestamp:yes /Version:no
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[How to: Use the Modules Window](#)

List Registers Command

12/22/2017 • 1 min to read • [Edit Online](#)

Displays the value of the selected registers and lets you modify the list of registers to show.

Syntax

```
Debug.ListRegisters [/Display [{register|registerGroup}...]] [/List]
[/Watch [{register|registerGroup}...]]
[/Unwatch [{register|registerGroup}...]]
```

Switches

/Display [{register | registerGroup}...]

Displays the values of the specified `register` or `registerGroup`. If no `register` or `registerGroup` is specified, the default list of registers is displayed. If no switch is specified, the behavior is the same. For example:

```
Debug.ListRegisters /Display eax
```

is equivalent to

```
Debug.ListRegisters eax
```

/List

Displays all register groups in the list.

/Watch [{register | registerGroup}...]

Adds one or more `register` or `registerGroup` values to the list.

/Unwatch [{register | registerGroup}...]

Removes one or more `register` or `registerGroup` values from the list.

Remarks

The alias `r` can be used in place of `Debug.ListRegisters`.

Example

This example uses the `Debug.ListRegisters` alias `r` to display the values of the register group `Flags`.

```
r /Display Flags
```

See Also

[Visual Studio Commands](#)

[Debugging Basics: Registers Window](#)

[How to: Use the Registers Window](#)

List Source Command

12/22/2017 • 1 min to read • [Edit Online](#)

Displays the specified lines of source code.

Syntax

```
Debug.ListSource [/Count:number] [/Current] [/File:filename]  
[/Line:number] [/ShowLineNumbers:yes|no]
```

Switches

/Count: `number`

Optional. Specifies the number of lines to display.

/Current

Optional. Shows the current line.

/File: `filename`

Optional. Path of the file to show. If no filename is specified, the command shows the source code for the line of the current statement.

/Line: `number`

Optional. Shows a specific line number.

/ShowLineNumbers: `yes|no`

Optional. Specifies whether to display line numbers.

Remarks

Example

This example lists the source code from line 4 of the file Form1.vb, with line numbers visible.

```
Debug.ListSource /File:"C:\Visual Studio Projects\Form1.vb" /Line:4 /ShowLineNumbers:yes
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

List Threads Command

12/22/2017 • 1 min to read • [Edit Online](#)

Displays a list of the threads in the current program.

Syntax

```
Debug.ListThreads [index]
```

Arguments

`index`

Optional. Selects a thread by its index to be the current thread.

Remarks

When specified, the `index` argument marks the indicated thread as the current thread. An asterisk (*) is displayed in the list next to the current thread.

Example

```
>Debug.ListThreads
```

See Also

[List Call Stack Command](#)

[List Disassembly Command](#)

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Log Command Window Output Command

12/22/2017 • 1 min to read • [Edit Online](#)

Copies all input and output from the **Command** window into a file.

Syntax

```
Tools.LogCommandWindowOutput [filename] [/on]/[off] [/overwrite]
```

Arguments

`filename`

Optional. The name of the log file. By default, the file is created in the user's profile folder. If the file name already exists, the log is appended to the end of the existing file. If no file is specified, the last file specified is used. If no previous file exists, a default log file is created, called cmdline.log.

TIP

To change the location where the log file is saved, enter the full path of the file, surrounded by quotation marks if the path contains any spaces.

Switches

`/on`

Optional. Starts the log for the **Command** window in the specified file and appends the file with the new information.

`/off`

Optional. Stops the log for the **Command** window.

`/overwrite`

Optional. If the file specified in the `filename` argument matches an existing file, the file is overwritten.

Remarks

If no file is specified, the file cmdline.log is created by default. By default, the alias for this command is Log.

Examples

This example creates a new log file, cmdlog, and starts the command log.

```
>Tools.LogCommandWindowOutput cmdlog
```

This example stops logging commands.

```
>Tools.LogCommandWindowOutput /off
```

This example resumes the logging of commands in the previously used log file.

```
>Tools.LogCommandWindowOutput /on
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

New File Command

12/22/2017 • 1 min to read • [Edit Online](#)

Creates a new file and opens it. The file appears under the Miscellaneous Files folder.

Syntax

```
File.NewFile [filename] [/t:templatename] [/editor:editorname]
```

Arguments

`filename`

Optional. Name for the file. If no name is supplied, a default name is provided. If no template name is listed, a text file is created.

Switches

`/t: templatename`

Optional. Specifies the type of file to be created.

The `/t: templatename` argument syntax mirrors the information found in the New File Dialog Box. Enter the category name followed by a backslash (`\`) and the template name, and enclose the entire string in quotation marks.

For example, to create a new Visual C++ source file, you would enter the following for the `/t: templatename` argument.

```
/t:"Visual C++\C++ File (.cpp)"
```

The example above indicates that the C++ File template is located under the Visual C++ category in the **New File** dialog box.

`/e: editorname`

Optional. Name of the editor in which the file will be opened. If the argument is specified but no editor name is supplied, the **Open With** dialog box appears.

The `/e: editorname` argument syntax uses the editor names as they appear in the Open With Dialog Box, enclosed in quotation marks.

For example, to open a file in the source code editor, you would enter the following for the `/e: editorname` argument.

```
/e:"Source Code (text) Editor"
```

Example

This example creates a new Web page "test1.htm" and opens it in the source code editor.

```
>File.NewFile test1 /t:"General\HTML Page" /e:"Source Code (text) Editor"
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Immediate Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Open File Command

12/22/2017 • 1 min to read • [Edit Online](#)

Opens an existing file and allows you to specify an editor.

Syntax

```
File.OpenFile filename [/e:editorname]
```

Arguments

`filename`

Required. The full or partial path and file name of the file to open. Paths containing spaces must be enclosed in quotation marks.

Switches

`/e: editorname`

Optional. Name of the editor in which the file will be opened. If the argument is specified but no editor name is supplied, the **Open With** dialog box appears.

The `/e: editorname` argument syntax uses the editor names as they appear in the Open With Dialog Box, enclosed in quotation marks.

For example, to open a file in the source code editor, you would enter the following for the `/e: editorname` argument.

```
/e:"Source Code (text) Editor"
```

Remarks

As you enter a path, auto completion tries to locate the correct path and file name.

Example

This example opens the style file "Test1.css" in the source code editor.

```
>File.OpenFile "C:\My Projects\project1\Test1.css" /e:"Source Code (text) Editor"
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Immediate Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Open Project Command

12/22/2017 • 1 min to read • [Edit Online](#)

Opens an existing project.

Syntax

```
File.OpenProject filename
```

Arguments

`filename`

Required. The full path and file name of the project to open.

The syntax for the `filename` argument requires that paths containing spaces use quotation marks.

Remarks

Auto completion tries to locate the correct path and file name as you type.

This command is not available while debugging.

Example

This example opens the Visual Basic project, Test1.

```
>File.OpenProject "C:\My Projects\Test1\Test1.vbproj"
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Open Solution Command

12/22/2017 • 1 min to read • [Edit Online](#)

Opens an existing solution, closing any other open solutions.

Syntax

```
File.OpenSolution filename
```

Arguments

Filename

Required. The full path and file name of the solution to open.

The syntax for the `filename` argument requires that paths containing spaces use quotation marks.

Remarks

Auto completion tries to locate the correct path and file name as you type.

Example

This example opens the solution, Test1.sln.

```
>File.OpenSolution "c:\MySolutions\Test1\Test1.sln"
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Print Command

12/22/2017 • 1 min to read • [Edit Online](#)

Evaluates an expression, or displays specified text.

Syntax

```
Debug.Print text
```

Arguments

`text`

Required. The expression to evaluate or the text to display.

Remarks

You can use the question mark (?) as an alias for this command. So, for example, the command

```
>Debug.Print expA
```

can also be written

```
>? expA
```

Both versions of this command will return the current value of the expression `expA`.

Example

```
>Debug.Print varA
```

See Also

[Evaluate Statement Command](#)

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Quick Watch Command

12/22/2017 • 1 min to read • [Edit Online](#)

Displays the selected or specified text in the Expression field of the [QuickWatch](#) window. You can use this dialog box to calculate the current value of a variable or expression recognized by the debugger, or the contents of a register. In addition, you can change the value of any non-const variable or the contents of any register.

Syntax

```
Debug.QuickWatchq [text]
```

Arguments

`text`

Optional. The text to add to the **QuickWatch** dialog box.

Remarks

If `text` is omitted, the currently selected text or word at the cursor is added to the Watch window.

Example

```
>Debug.QuickWatch
```

See Also

[Set a Watch on Variables using the Watch and QuickWatch Windows in Visual Studio](#)

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Replace Command

12/22/2017 • 1 min to read • [Edit Online](#)

Replaces text in files using a subset of the options available on the **Replace in Files** tab of the **Find and Replace** window.

Syntax

```
Edit.Replace findwhat replacewith [/all] [/case]
[/doc]/[proc]/[open]/[sel] [/hidden] [/options] [/reset] [/up]
[/wild]/[regex] [/word]
```

Arguments

`findwhat`

Required. The text to match.

`replacewith`

Required. The text to substitute for the matched text.

Switches

`/all` or `/a`

Optional. Replaces all occurrences of the search text with the replacement text.

`/case` or `/c`

Optional. Matches occur only if when the uppercase and lowercase characters exactly match those specified in the `findwhat` argument.

`/doc` or `/d`

Optional. Searches the current document only. Specify only one of the available search scopes, `/doc`, `/proc`, `/open`, or `/sel`.

`/hidden` or `/h`

Optional. Searches concealed and collapsed text, such as the metadata of a design-time control, a hidden region of an outlined document, or a collapsed class or method.

`/open` or `/o`

Optional. Searches all open documents as if they were one document. Specify only one of the available search scopes, `/doc`, `/proc`, `/open`, or `/sel`.

`/options` or `/t`

Optional. Displays a list of the current find option settings and does not perform a search.

`/proc` or `/p`

Optional. Searches the current procedure only. Specify only one of the available search scopes, `/doc`, `/proc`, `/open`, or `/sel`.

`/regex` or `/r`

Optional. Uses pre-defined special characters in the `findwhat` argument as notations that represent patterns of text rather than the literal characters. For a complete list of regular expression characters, see [Regular Expressions](#).

/reset or /e

Optional. Returns the find options to their default settings and does not perform a search.

/sel or /s

Optional. Searches the current selection only. Specify only one of the available search scopes, `/doc`, `/proc`, `/open`, or `/sel`.

/up or /u

Optional. Searches from the current location in the file toward the top of the file. By default, searches begin at the current location in the file and advance toward the bottom of the file.

/wild or /l

Optional. Uses pre-defined special characters in the `findwhat` argument as notations to represent a character or sequence of characters.

/word or /w

Optional. Searches only for whole words.

Example

This example replaces `btnSend` with `btnSubmit` in all open documents.

```
>Edit.Replace btnSend btnSubmit /open
```

See Also

[Finding and Replacing Text](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Commands](#)

[Visual Studio Command Aliases](#)

Replace In Files Command

12/22/2017 • 1 min to read • [Edit Online](#)

Replaces text in files using a subset of the options available on the **Replace in Files** tab of the **Find and Replace** window.

Syntax

```
Edit.ReplaceInFiles findwhat replacewith [/all] [/case]
[/ext:extensions] [/keep] [/lookin:searchpath] [/options] [/regex]
[/reset] [/stop] [/sub] [/text2] [/wild] [/word]
```

Arguments

`findwhat`

Required. The text to match.

`replacewith`

Required. The text to substitute for the matched text.

Switches

`/all` or `/a`

Optional. Replaces all occurrences of the search text with the replacement text.

`/case` or `/c`

Optional. Matches occur only if when the uppercase and lowercase characters exactly match those specified in the `findwhat` argument.

`/ext: extensions`

Optional. Specifies the file extensions for the files to be searched.

`/keep` or `/k`

Optional. Specifies that all modified files are left open.

`/lookin: searchpath`

Optional. Directory to search. If the path contains spaces, enclose the entire path in quotation marks.

`/options` or `/t`

Optional. Displays a list of the current find option settings and does not perform a search.

`/regex` or `/r`

Optional. Uses pre-defined special characters in the `findwhat` argument as notations that represent patterns of text rather than the literal characters. For a complete list of regular expression characters, see [Regular Expressions](#).

`/reset` or `/e`

Optional. Returns the find options to their default settings and does not perform a search.

`/stop`

Optional. Halts the current search operation if one is in progress. Replace ignores all other arguments when `/stop` has been specified. For example, to stop the current replacement you would enter the following:

```
>Edit.ReplaceinFiles /stop
```

/sub or /s

Optional. Searches the subfolders within the directory specified in the /lookin: `searchpath` argument.

/text2 or /2

Optional. Displays the results of the replacement in the **Find Results 2** window.

/wild or /l

Optional. Uses pre-defined special characters in the `findwhat` argument as notations to represent a character or sequence of characters.

/word or /w

Optional. Searches for only whole words.

Example

This example searches for `btnCancel` and replaces it with `btnReset` in all .cls files located in the folder "my visual studio projects" and displays the replacement information in the **Find Results 2** window.

```
>Edit.ReplaceinFiles btnCancel btnReset /lookin:"c:/my visual studio projects" /ext:.cls /text2
```

See Also

[Finding and Replacing Text](#)

[Replace in Files](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Commands](#)

[Visual Studio Command Aliases](#)

Set Current Process

12/22/2017 • 1 min to read • [Edit Online](#)

Sets the specified process as the active process in the debugger.

Syntax

```
Debug.SetCurrentProcess index
```

Arguments

`index`

Required. The index of the process.

Remarks

You can attach to multiple processes when you are debugging, but only one process is active in the debugger at any given time. You can use the `SetCurrentProcess` command to set the active process.

Example

```
>Debug.SetCurrentProcess 1
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Visual Studio Command Aliases](#)

Set Current Stack Frame Command

12/22/2017 • 1 min to read • [Edit Online](#)

Allows you to set a particular stack frame.

Syntax

```
Debug.SetCurrentStackFrame index
```

Arguments

`index`

Required. Selects a stack frame by its index.

Example

```
>Debug.SetCurrentStackFrame 1
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Set Current Thread Command

12/22/2017 • 1 min to read • [Edit Online](#)

Sets the specified thread as the current thread.

Syntax

```
Debug.SetCurrentThread index
```

Arguments

`index`

Required. Selects a thread by its index.

Example

```
>Debug.SetCurrentThread 1
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Set Radix Command

12/22/2017 • 1 min to read • [Edit Online](#)

Sets or returns the numeric base used to display integer values.

Syntax

```
Debug.SetRadix [10 | 16 | hex | dec]
```

Arguments

`10` or `16` or `hex` or `dec`

Optional. Indicates decimal (10 or dec) or hexadecimal (16 or hex). If an argument is omitted, then the current radix value is returned.

Example

This example sets the environment to display integer values in hexadecimal format.

```
>Debug.SetRadix hex
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Shell Command

12/22/2017 • 1 min to read • [Edit Online](#)

Launches executable programs from within Visual Studio.

Syntax

```
Tools.Shell [/command] [/output] [/dir:folder] path [args]
```

Arguments

`path`

Required. The path and file name of the file to execute or the document to open. A full path is required if the specified file is not in one of the directories in the PATH environment variable.

`args`

Optional. Any arguments to pass to the invoked program.

Switches

`/commandwindow [or] /command [or] /c [or] /cmd`

Optional. Specifies that the output for the executable is displayed in the **Command** window.

`/dir: folder [or] /d: folder`

Optional. Specifies the working directory to be set when the program is run.

`/outputwindow [or] /output [or] /out [or] /o`

Optional. Specifies that the output for the executable is displayed in the **Output** window.

Remarks

The `/dir` `/o` `/c` switches must be specified immediately after `Tools.Shell`. Anything specified after the name of the executable is passed to it as command line arguments.

The predefined alias `Shell` can be used in place of `Tools.Shell`.

Caution

If the `path` argument supplies the directory path as well as the file name, you should enclose the entire pathname in literal quotes (""""), as in the following:

```
Tools.Shell """C:\Program Files\SomeFile.exe"""
```

Each set of three double quotes (""""") is interpreted by the `Shell` processor as a single double quote character. Thus, the preceding example actually passes the following path string to the `Shell` command:

```
"C:\Program Files\SomeFile.exe"
```

Caution

If you do not enclose the path string in literal quotes (""""), Windows will use only the portion of the string up to the first space. For example, if the path string above were not quoted properly, Windows would look for a file named

"Program" located in the C:\ root directory. If a C:\Program.exe executable file were actually available, even one installed by illicit tampering, Windows would attempt to execute that program in place of the desired "c:\Program Files\SomeFile.exe" program.

Example

The following command uses xcopy.exe to copy the file MyText.txt into the Text folder. The output from xcopy.exe is displayed in both the **Command Window** and the **Output** window.

```
>Tools.Shell /o /c xcopy.exe c:\MyText.txt c:\Text\MyText.txt
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Output Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

ShowWebBrowser Command

12/22/2017 • 1 min to read • [Edit Online](#)

Displays the URL you specify in a Web browser window either within the integrated development environment (IDE) or external to the IDE.

Syntax

```
View.ShowWebBrowser URL [/new][/ext]
```

Arguments

URL

Required. URL (Uniform Resource Locator) for the Web site.

Switches

/new

Optional. Specifies that the page appears in a new instance of the Web browser.

/ext

Optional. Specifies that the page appears in the default Web browser outside of the IDE.

Remarks

The alias for the **ShowWebBrowser** command is **navigate** or **nav**.

Example

The following example displays the MSDN Online home page in a Web browser outside of the IDE. If an instance of the Web browser is already open, it is used; otherwise a new instance is launched.

```
>View.ShowWebBrowser http://msdn.microsoft.com /ext
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Start Command

12/22/2017 • 1 min to read • [Edit Online](#)

Begins debugging the startup project.

Syntax

```
Debug.Start [address]
```

Arguments

address

Optional. The address at which the program suspends execution, similar to a breakpoint in source code. This argument is only valid in debug mode.

Remarks

The **Start** command, when executed, performs a RunToCursor operation to the specified address.

Example

This example starts the debugger and ignores any exceptions that occur.

```
>Debug.Start
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Symbol Path Command

12/22/2017 • 1 min to read • [Edit Online](#)

Sets the list of directories for the debugger to search for symbols.

Syntax

```
Debug.SymbolPath pathname1;pathname2;... pathnameN
```

Arguments

`pathname`

Optional. A semi-colon delimited list of paths for the debugger to search for symbols.

Remarks

If no `pathname` is specified, the command lists the current symbol paths.

Example

This example adds two paths to the list of symbol directories.

```
Debug.SymbolPath C:\Symbol Path 1;C:\Symbol Path 2
```

Example

This example displays a semi-colon delimited list of current symbol paths.

```
Debug.SymbolPath
```

See Also

[Command Window](#)

[Visual Studio Commands](#)

Toggle Breakpoint Command

12/22/2017 • 1 min to read • [Edit Online](#)

Turns the breakpoint either on or off, depending on its current state, at the current location in the file.

Syntax

```
Debug.ToggleBreakpoint [text]
```

Arguments

text

Optional. If text is specified, the line is marked as a named breakpoint. Otherwise, the line is marked as an unnamed breakpoint, which is similar to what happens when you press F9.

Example

The following example toggles the current breakpoint.

```
>Debug.ToggleBreakpoint
```

See Also

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Watch Command

12/22/2017 • 1 min to read • [Edit Online](#)

Creates and opens a specified instance of a **Watch** window. You can use a **Watch** window to calculate the values of variables, expressions, and registers, to edit these values, and to save the results.

Syntax

```
Debug.Watch[index]
```

Arguments

`index`

Required. The instance number of the watch window.

Remarks

The `index` must be an integer. Valid values are 1, 2, 3, or 4.

Example

```
>Debug.Watch1
```

See Also

[Autos and Locals Windows](#)

[Set a Watch on Variables using the Watch and QuickWatch Windows in Visual Studio](#)

[Visual Studio Commands](#)

[Command Window](#)

[Find/Command Box](#)

[Visual Studio Command Aliases](#)

Microsoft Help Viewer

12/22/2017 • 3 min to read • [Edit Online](#)

You can install and view content for various products and technologies on your local computer by using Microsoft Help Viewer, including Visual Studio, the .NET Framework, language reference, SQL Server, and Windows Development. Help Viewer enables you to:

- Find and download sets of content, which are also referred to as books.
- Find topics by title by browsing and searching the table of contents.
- Look up subjects in the index.
- Find information by using full-text search.
- View, bookmark, and print topics.

To install Help Viewer, see [Microsoft Help Viewer Installation](#). To start reading help topics in the Help Viewer rather than online, go to the **Help** menu in Visual Studio, and then choose **Set Help Preference, Launch in Help Viewer**.

Help Viewer tour

You can find information in installed content by using the navigation tabs, view installed content in the topic tab or tabs, and manage content by using the **Manage Content** tab. You can also perform additional tasks by using the buttons on the toolbar and find additional information in the lower-right corner of the window.

Navigation tabs

| TAB | DESCRIPTION |
|-----------|--|
| Contents | Displays installed content as a hierarchy (table of contents). You can specify criteria to filter the titles that appear. |
| Index | Displays an alphabetical list of indexed terms. You can search the index, specify criteria to filter the entries, and require that index entries either contain or start with text that you specify. |
| Favorites | You can "favorite" topics by choosing the Add to Favorites button, and the topics appear in this tab. The History section displays a list of topics that you've viewed recently. |
| Search | Provides a text box where you can search for terms anywhere in the content, including code and topic titles. |

Viewing topics

Each topic appears in its own tab, and you can open multiple topics at the same time.

Managing content

You can install, update, move, and delete content by using the **Manage Content** tab. At the top of the tab, you can use the **Installation source** control to specify whether to install books from a network location or from a disk or URI. The **Local store path** box shows where books are installed on the local computer, and you can move them to a different location by choosing the **Move** button.

The content list shows which books you can install or have already installed, whether an update is available, and how large each book is. You can install or remove one or more books by choosing the appropriate **Add** or **Remove** links and then choosing the **Update** button under the **Pending changes** pane. If updates are available for any books that you've already installed, you can refresh that content by choosing the **Click here to download now** link at the bottom of the window. In addition, all installed books are refreshed if updates are available when you install additional books.

Note: The functionality of the **Manage Content** tab may differ if the Help Viewer administrator deactivates these features, or if no internet access is available.

Toolbar buttons

The toolbar in the Help Viewer window contains the following buttons:

- The **Show Topic in Contents** button shows the location of the topic in the **Contents** tab.
- The **Add to Favorites** button adds the active topic to the **Favorites** tab.
- The **Find in Topic** button highlights search text in the active topic.
- The **Print** button prints or shows a preview of the active topic.
- The **Viewer Options** button displays settings such as how large the text appears, how many search results to return, how many topics to show in history, and whether to check for updates online.
- The **Manage Content** button makes the **Manage Content** tab active.
- The small triangle on the right-hand side opens a list of tabs, including topic tabs and the **Manage Content** tab. You can choose a tab name to make it the active tab.

See also

[Microsoft Help Viewer Installation](#)

[Help Viewer Administrator Guide](#)

[Install and Manage Local Content](#)

Install and manage local content

12/22/2017 • 3 min to read • [Edit Online](#)

By using the Microsoft Help Viewer, you can add, remove, update, and move the Help content that is installed on your computer to fit your software development needs.

To manage content on your local computer, you must log on with an account that has administrator permissions. In addition, you might not be able to manage local content if you work in an enterprise environment, because system administrators might make those decisions for your organization. For more information, see the [Help Viewer Administrator Guide](#).

Changing the content installation source

By default, the Help Viewer installs content by using a Microsoft online service as the source. You generally shouldn't change your content source unless you work in an enterprise environment for which a system administrator has already installed content in another location.

To change the content installation source

1. On the **Manage Content** tab, choose the **Disk** option button.

NOTE

The **Disk** option isn't available if your administrator has prevented you from modifying the content installation source. For more information, see the [Help Viewer Administrator Guide](#).

2. Perform one of the following steps:

- Enter the path of an .msha file or the URL of a service endpoint.
- Choose the **Browse (...)** button to navigate to an .msha file.
- In the list, choose the entry that was used most recently.

Download and install content locally

If you download and install content on your local computer, you can view topics when you don't have an internet connection.

IMPORTANT

To install content, you must log on with an account that has administrative permissions.

NOTE

If the Visual Studio IDE is set to a language other than English, you can install English content, localized content, or both. However, no content appears if you install only the English version and the **Include English content in all navigation tabs and F1 requests** check box in the **Viewer Options** dialog box is cleared.

To download and install content

1. Choose the **Manage Content** tab.

2. In the content list, choose the **Add** link next to the book or books that you want to download and install.

The book is added to the **Pending changes** list, and the estimated size of the book or books that you specified appears below that list. Because some books share topics, the total download size of multiple books might be smaller than the result of adding together the sizes of every book that you specified.

3. Choose the **Update** button.

The book or books that you specified are installed along with any updates for books that you already have on your computer. Installation times vary, but you can view the progress in the status bar.

Removing local content

You can save disk space by removing unwanted content from your computer.

IMPORTANT

You must have administrative permissions to remove content.

NOTE

No content appears if the Visual Studio IDE is set to a language other than English, you remove localized content, and the **Include English content in all navigation tab and F1 requests** check box in the **Viewer Options** dialog box is cleared.

To remove content

1. Choose the **Manage Content** tab.
2. In the content list, choose the **Remove** link next to the book or books that you want to remove.

The book is added to the **Pending changes** list.

3. Choose the **Update** button.

The book or books that you specified are removed from your computer.

Updating local content

The status bar indicates when updates to your installed content are available.

IMPORTANT

If you want the Help Viewer to automatically check for online updates, you must open the **Viewer Options** dialog box and then select the **Go online to check for content updates** check box.

To update local content

- In the lower-right corner of the status bar, choose the **Click here to download now** link.

Update times can vary, but you can view the update progress in the status bar.

Moving local content

You can save disk space by moving installed content from your local computer to a network share or to another partition on your local computer.

IMPORTANT

To move content, you must log on with an account that has administrative permissions.

To move local content

1. On the **Manage Content** tab, choose the **Move** button under **Local Store Path**.

The **Move Content** dialog box opens.

2. In the **To** text box, enter a different location for the content, and then choose the **OK** button.
3. Choose the **Close** button when the content has been moved.

See also

[Microsoft Help Viewer](#)

Finding topics by using the Help Viewer index

12/22/2017 • 2 min to read • [Edit Online](#)

The index contains a list of keywords that are associated with topics in the installed content. Each topic might have more than one keyword associated with it, and each keyword might be associated with more than one topic. Use this index in the same way as you would use an index in a book.

To find a topic by using the index

On the **Index** tab, perform either of the following tasks:

- Specify the keyword to search for in the text box. For example, specify "update" to find topics with keywords such as "update," "updated," and "updating."

By choosing the filter button near the top of the tab, you can display either all entries that contain the text that you specify or only those entries that start with the text that you specify.

NOTE

When the filter button appears on a darker background with a border, entries must *contain* the text that you specify. If the background and border don't appear, entries must *start with* the text that you specify.

- Scroll through the index, and choose a keyword.

If the keyword that you specify is associated with only one topic, it appears. Otherwise, a list of all topics that are associated with that keyword appears.

Index search tips

Using the index is a straight-forward process; however, understanding how to best enter keywords can make your index searches more productive.

General guidelines

- Scroll through the index entries. Not all topics are indexed the same way, and the one that could most help you might be higher or lower in the list than you expected.
- Omit articles such as "an" or "the" because the index ignores them.
- Reverse the words you enter if you do not find the entries you expect.

For example, if "debugging inline assembly code" did not display any relevant entries, try typing, "assembly code, debugging inline".

- Use filters with the **Index** tab to decrease the number of results.

Syntax tips

If you do not find an entry for the word or phrase you entered, try the following:

- Type the first few letters, or root, of the word. By entering a partial string, you can get to topics that have been indexed with keywords that are singular or plural.

For example, enter "propert" to start your search above properties and property.

- Enter gerund (-ing) forms of the verb for the task you want to complete. To find more specific index entries,

append a word that describes exactly what you want.

For example, type "running" to get more entries or "running programs" to get fewer.

- Enter standalone adjectives. To narrow the results, append a word that describes exactly what you want.

For example, enter "COM+" to get a wide range of entries or "COM+ components" to get fewer.

- Enter a synonym of the word or verb you are looking for.

For example, if you entered the term "building", try "creating" instead.

See also

[How to: find topics in the TOC](#)

[How to: search for topics](#)

[Microsoft Help Viewer](#)

How to: find topics in the table of contents

12/22/2017 • 1 min to read • [Edit Online](#)

In the **Contents** tab, you can use the table of contents (TOC) to find information. The table of contents is an expandable list that contains all of the topics in the installed books. For accessibility information about how to navigate through the TOC, see [Shortcut Keys \(Help Viewer\)](#).

IMPORTANT

The scope of topics available in the TOC depends on the filter you have selected.

Filter the TOC

You can filter the TOC to narrow the scope of topics that appear in the **Contents** tab. Titles appear in the list only if they contain the root of the term that you specify. For example, if you specify "troubleshooting" as a filter, only titles that contain "troubleshoot" or "troubleshooting" appear. Nodes whose titles don't contain the term are collapsed to a single node with an ellipsis (...).

To filter the TOC

1. Choose the **Contents** tab.
2. In the **Filter Contents** text box, enter a term.

NOTE

If the filter takes a long time to run, you might display results more quickly by using the `title:` advanced search operator.

Synchronize a topic with the TOC

If you have opened a topic using the index or full-text search features, you can determine where this topic is in the TOC by synchronizing the TOC with the topic window.

To synchronize the TOC with the topic window

1. View a topic.
2. Click the **Show Topic in Contents** button on the toolbar, or press **Ctrl+S**.

The **Contents** tab opens and displays the topic's location in the TOC.

See also

[How to: find topics in the index](#)

[How to: search for topics](#)

[Microsoft Help Viewer](#)

How to: search for topics

12/22/2017 • 3 min to read • [Edit Online](#)

You can use the full-text search feature to locate all topics that contain a particular word. You can also refine and customize your search by using wildcard expressions, logical operators, and advanced search operators.

To open the Search tab, choose the **Search** tab in the Help Viewer window, or if you are a keyboard user, choose **Ctrl+E**.

To perform a full-text search

1. In the search box, type the word that you want to find.
2. In the search query, specify which logical or advanced search operators to apply to the search, if any. To search all available Help, don't use operators.

NOTE

In the **Viewer Options** dialog box, you can specify additional preferences such as the maximum number of search results to display at a time and whether to include English content if your primary locale is not English.

3. Choose the **Enter** key.

A search returns a maximum of 200 hits, by default, and displays them in the search results area. Additional version information for each result may appear, depending on the content.

4. To view a topic, choose its title from the results list.

Full-text search tips

You can create more targeted searches that return only those topics that interest you, if you understand how syntax affects your query. The syntax includes special characters, reserved words, and filters. This topic provides tips, procedures, and detailed syntax information to help you better craft your queries.

General guidelines

The following table includes some basic rules and guidelines for developing search queries in Help.

| SYNTAX | DESCRIPTION |
|------------------------|---|
| Case sensitivity | Searches aren't case-sensitive. Develop your search criteria using uppercase or lowercase characters. For example, "OLE" and "ole" return the same results. |
| Character combinations | You can't search only for individual letters (a-z) or numbers (0-9). If you try to search for certain reserved words, such as "and", "from", and "with", they will be ignored. For more information, see Words ignored in searches later in this topic. |
| Evaluation order | Search queries are evaluated from left to right. |

Search syntax

If you specify a search string that includes multiple words, such as "word1 word2," that string is equivalent to

typing "word1 AND word2", which returns only topics that contain all of the individual words in the search string.

IMPORTANT

- Phrase searches are not supported. If you specify more than one word in a search string, returned topics will contain all of the words that you specified but not necessarily the exact phrase that you specified.
- Use logical operators to specify the relationship between words in your search phrase. You can include logical operators, such as AND, OR, NOT, and NEAR, to further refine your search. For example, if you search for "declaring NEAR union", search results will include topics that contain the words "declaring" and "union" no more than a few words apart from each other. For more information, see [Logical Operators in Search Expressions](#).

Filters

You can further restrict search results by using advanced search operators. Help includes three categories that you can use to filter results of a full-text search: Title, Code, and Keyword.

Ranking of search results

The search algorithm applies certain criteria to help rank search results higher or lower in the results list. In general:

1. Content that includes search words in the title is ranked higher than content that doesn't.
2. Content that includes search words in close proximity is ranked higher than content that doesn't.
3. Content that contains a higher density of the search words is ranked higher than content that has a lower density of the search words.

Words ignored in searches (stop words)

Commonly occurring words or numbers, which are sometimes called stop words, are automatically ignored during a full-text search. For example, if you search for the phrase "pass through", search results will display topics that contain the word "pass" but not "through".

See also

[Logical and advanced operators](#)

[How to: find topics in the index](#)

[How to: find topics in the TOC](#)

[Microsoft Help Viewer](#)

Logical and advanced operators in search expressions

12/22/2017 • 1 min to read • [Edit Online](#)

You can use logical operators and advanced search operators to refine your search of the Help content in Help Viewer.

Logical operators

Logical operators specify how multiple search terms should be combined in a search query. The following table shows the logical operators AND, OR, NOT and NEAR.

| TO SEARCH FOR | USE | EXAMPLE | RESULT |
|--|------|----------------------------|--|
| Both terms in the same article | AND | dib AND palette | Topics that contain both "dib" and "palette". |
| Either term in an article | OR | raster OR vector | Topics that contain either "raster" or "vector". |
| First term without the second term in the same article | NOT | "operating system" NOT DOS | Topics that contain "operating system" but not "DOS". |
| Both terms, close together in an article | NEAR | user NEAR kernel | Topics that contain "user" within close proximity of "kernel". |

IMPORTANT

You must enter logical operators in all capital letters for the search engine to recognize them.

Advanced operators

Advanced search operators refine your search for content by specifying where in an article to look for the search term. The following table describes the four available advanced search operators.

| TO SEARCH FOR | USE | EXAMPLE | RESULT |
|---|----------|--------------------|--|
| A term in the title of the article | title: | title:binaryreader | Topics that contain "binaryreader" in their titles. |
| A term in a code example | code: | code:readdouble | Topics that contain "readdouble" in a code example. |
| A term in an example of a specific programming language | code:vb: | code:vb:string | Topics that contain "string" in a Visual Basic code example. |

| TO SEARCH FOR | USE | EXAMPLE | RESULT |
|---|----------|------------------|---|
| An article that is associated with a specific index keyword | keyword: | keyword:readbyte | Topics that are associated with the "readbyte" index keyword. |

IMPORTANT

You must enter advanced search operators with a final colon and no intervening space before the colon for the search engine to recognize them.

Programming languages for code examples

You can use the **code:** operator to find content about any of several programming languages. To return examples for a specific programming language, use one of the following programming language values:

| PROGRAMMING LANGUAGE | SEARCH OPERATOR SYNTAX |
|----------------------|--|
| Visual Basic | code:vb code:visualbasic |
| C# | code:c# code:csharp |
| C++ | code:cpp code:c++ code:cplusplus |
| F# | code:f# code:fsharp |
| JavaScript | code:javascript code:js |
| XAML | code:xaml |

NOTE

The **code:** operator only finds content that is marked up with a programming language label, as opposed to content that is generically marked up as code.

See

- [How to: search for topics](#)
- [Microsoft Help Viewer](#)

Customize the help viewer

12/22/2017 • 1 min to read • [Edit Online](#)

You can customize the layout of the Help Viewer windows, as well as other options such as font size, maximum number of results, and whether to include English content.

Customizing window layout

You can customize the window layout of the Help Viewer. To restore the Help Viewer window to its default layout, open the **Viewer Options** dialog box, and then choose the **Reset** button.

Docking tabs

The Help Viewer supports standard docking functionality. By default, all tabs in the Help Viewer are docked, but you can move them, resize them, dock them in other locations, and "float" them so that they appear as independent child windows.

Opening a topic in a new tab

Choose the topic in any navigation tab, and then press **Ctrl+Enter**.

Minimize a navigation tab

Create more space for viewing topics by choosing the pin icon for the navigation tabs. When these tabs are minimized, only their labels appear on the closest edge of the window. To restore the tabs, choose the label of any tab, and then choose the pin icon again.

Changing settings in Viewer Options

You open the **Viewer Options** dialog box by choosing the **Viewer Options** button on the toolbar.

| TO PERFORM THIS TASK: | TAKE THIS STEP: |
|--|--|
| Change the size of the font in which text appears | Choose a size in the Text Size list. |
| Change the maximum number of search results that appear in the Search tab | Choose a value in the Maximum Search Results list. |
| Change the maximum number of history entries that appear in the Favorites window | Choose a value in the Maximum History entries saved list. |
| Include or exclude English content when you view content for a non-English version of a product. | Select or clear the Include English content in all navigation tabs and F1 requests check box. Caution: This feature also controls whether you can download English content in the Manage Content tab. |

See also

[Microsoft Help Viewer](#)

Accessibility features of the Help Viewer

12/22/2017 • 1 min to read • [Edit Online](#)

Microsoft is committed to making its products and services easier for everyone to use. This topic includes information about the features, products, and services that help make Microsoft Help Viewer accessible for people with a wide range of abilities.

Keyboard access

You can access all features of the Help Viewer by using the keyboard. For more information, see [Shortcut Keys \(Help Viewer\)](#).

Font size

You can modify the font size in which topic text appears in the document window. For more information, see [Customize the Help Viewer](#).

Window size

You can change the width of the navigation or document windows by pointing to the divider between the two windows. When the cursor changes to a double-headed arrow, use the primary mouse button to drag the divider to the right or left.

Help Viewer position

You can reposition the Help Viewer by dragging its title bar to a different position.

See also

[Microsoft Help Viewer](#)

[Shortcut Keys \(Help Viewer\)](#)

Shortcut Keys (Help Viewer)

12/22/2017 • 5 min to read • [Edit Online](#)

You can navigate in the Microsoft Help Viewer by using the shortcut keys in the following table:

| AREA | KEYSTROKE | ACTION |
|---------------------|---------------------------------------|--|
| General Application | Space | Use instead of Enter anywhere except in edit fields. |
| General Application | F1 | Open Help about current UI element. |
| General Application | F11 | Toggle between full-screen view and regular view. |
| Toolbar | Backspace -OR- Alt + Left Arrow | Display the previous page. |
| Toolbar | Alt + Right Arrow | Display the next page. |
| Toolbar | Alt + Home | Display the Help Reviewer Home page. |
| Toolbar | Ctrl + S | Highlight the current topic in the table of contents (on the Contents tab). |
| Toolbar | Ctrl + D | Add the current topic to the Favorites tab. |
| Toolbar | Ctrl + F | Display the Find bar in the topic area so that you can search for text within the current topic. |
| Toolbar | Ctrl + P | Print the current page. |
| Toolbar | Ctrl + F2 | Display a print preview of the current page. |
| Toolbar | Ctrl + O | Display the Viewer Options dialog box. |
| Toolbar | Ctrl + Shift + M | Display the Manage Contents tab. |
| Navigators | Alt + C -OR- Ctrl + Shift + C | Display the Contents tab. |

| AREA | KEYSTROKE | ACTION |
|------------|---|---|
| Navigators | Alt + I -OR- Ctrl + Shift + I | Display the Index tab. |
| Navigators | Alt + F -OR- Ctrl + Shift + F | Display the Favorites tab. |
| Navigators | Alt + S -OR- Ctrl + E -OR- Ctrl + Shift + S | Display the Search tab. |
| Navigators | Alt + M -OR- Ctrl + Shift + M | Display the Manage Content tab. |
| Topic | Shortcut Menu key OR Shift + F10 | Display the shortcut menu for the current UI element. |
| Topic | Up Arrow | Scroll toward the start of the document one line at a time. |
| Topic | Down Arrow | Scroll toward the end of the document one line at a time. |
| Topic | Page Up | Scroll toward the start of the document one screen at a time. |
| Topic | Page Down | Scroll toward the end of the document one screen at a time. |
| Topic | Home | Move to the start of the document. |
| Topic | End | Move to the end of the document. |
| Topic | Ctrl + F | Find search text on this page. |
| Topic | F5 | Refresh the current page. |
| Topic | Ctrl + P | Print the current page. |

| AREA | KEYSTROKE | ACTION |
|---------------------------------------|--------------------------|--|
| Topic | Ctrl + F2 | Display a print preview of the current page. |
| Topic | F4 | Display the Properties dialog box for the current page. |
| Topic | Ctrl + T | Open another content tab in the foreground. |
| Topic | Ctrl + Click | Open a link on a new tab in the foreground. |
| Topic | Ctrl + Tab | Switch among tabs from left to right. |
| Topic | Ctrl + Shift + Tab | Switch among tabs from right to left. |
| Topic | Ctrl + W | Close the current tab. |
| Topic | Ctrl + Number | Switch to a specific tab where <i>Number</i> is between 1 and 9 and indicates which tab in sequence. |
| Topic | Ctrl + Alt + F4 | Close other content tabs. |
| Topic | Ctrl + Shift + Plus Sign | Increase zoom by 10%. |
| Topic | Ctrl + Minus Sign | Decrease zoom by 10%. |
| Topic | Ctrl + 0 (zero) | Change zoom to 100%. |
| Index | Tab | Shift focus from keyword entry to keyword list. |
| Index | Ctrl + K | Switch between showing entries that contain the keyword that you specify and entries that start with the keyword that you specify. |
| Favorites | Ctrl + Shift + Del | Clear your browsing history. |
| Favorites | Del | Delete the specified item. |
| Favorites | Ctrl + N | Create a folder within Favorites. |
| Favorites | F2 | Rename the specified favorite or folder. |
| Contents & Index & Search | Ctrl + D | Add the specified topic to the Favorites tab. |
| Contents & Index & Search & Favorites | Ctrl + P | Print the specified topic. |

| AREA | KEYSTROKE | ACTION |
|---------------------------------------|---------------|---|
| Contents & Index & Search & Favorites | Ctrl + F2 | Display a print preview of the specified topic. |
| Contents & Index & Search & Favorites | Ctrl + Click | Open the topic in a new tab. |
| Search | Esc | Clear the search text box. |
| Viewer Options | Alt + T | Change focus to the Text Size list. |
| Viewer Options | Alt + S | Change focus to the Maximum Search Results list. |
| Viewer Options | Alt + H | Change focus to the Maximum History entries saved list. |
| Viewer Options | Alt + E | Select or clear the Include English content in all navigation tabs and F1 requests check box if it is enabled. |
| Viewer Options | Alt + O | Select or clear the Go online to check for content updates check box. |
| Find | Enter | Change focus to the next item. |
| Find | Shift + Enter | Change focus to the previous item. |
| Find | Esc | Hides the Find text box. |
| Status bar | Alt + E | Open the Error dialog box if the status bar shows that an error has occurred. |
| Status bar | Alt + U | Download content if the status bar shows that updates are available |

Window Management

| Keystroke | Action |
|--------------------|--|
| Ctrl + L | Reset the Help Viewer layout to the default layout, and close all topic tabs. |
| Ctrl + Tab | The first keystroke gives focus to the Tab Selection menu. The next keystroke gives focus to the top menu item, and subsequent keystrokes give focus to the menu items in sequence from top to bottom. When a menu item has focus, the Enter key makes that item the active tab. |
| Ctrl + Shift + Tab | The first keystroke gives focus to the Tab Selection menu. The next keystroke gives focus to the bottom menu item, and subsequent keystrokes give focus to the menu items in sequence from bottom to top. When a menu item has focus, the Enter key makes that item the active tab. |

| | |
|---|--|
| Alt + I, Alt + S, Alt + C, Alt + F, Alt + M | These shortcut keys don't work when the navigation and content-management tabs are undocked. |
|---|--|

Manage Content

| Keystroke | Action |
|-------------------------|---|
| Alt + D | Change the installation source to disk. |
| Alt + O | Change the installation source to online. |
| Tab | Change focus to the Local store path text box. |
| Tab | Change focus to the Move... button. |
| Alt + V | Open the Move Content dialog box. |
| Ctrl + Alt + F | Change focus to the Filter Documentation text box. |
| Tab | Change focus to the documentation list. |
| Up Arrow and Down Arrow | Scroll through the documentation list. |
| Space | Add an item to the Pending changes list. |
| Tab | Change focus to the Pending changes list. |
| Up Arrow and Down Arrow | Scroll through the Pending changes list. |
| Space | Remove an item from the Pending changes list. |
| Alt + T | Apply all pending changes. |

See also

[Accessibility features of the Help Viewer](#)

Dotfuscator Community Edition (CE)

10/31/2017 • 3 min to read • [Edit Online](#)

PreEmptive Protection - Dotfuscator provides comprehensive .NET application protection that easily fits into your secure software development lifecycle. Use it to harden, protect, and prune desktop, mobile, server, and embedded applications to help secure trade secrets and other intellectual property (IP), reduce piracy and counterfeiting, and protect against tampering and unauthorized debugging. Dotfuscator works on compiled assemblies without the need for additional programming or even access to source code.

PreEmptive Protection



Why Protection Matters

It's important to **protect your intellectual property** (IP). Your application's code contains design and implementation details which can be considered IP. However, applications built on the .NET Framework **contain significant metadata and high-level intermediate code**, making them very easy to reverse engineer, just by using one of many free, automated tools. By disrupting and stopping reverse-engineering, you can prevent unauthorized IP disclosure, as well as demonstrate that your code contains trade secrets. Dotfuscator can **obfuscate** your .NET assemblies to hinder reverse-engineering, while maintaining original application behavior.

It's also important to **protect the integrity of your application**. In addition to reverse-engineering, bad actors may attempt to pirate your application, alter the application's behavior at runtime, or manipulate data. Dotfuscator can inject your application with the capability to **detect, report, and respond to unauthorized uses**, including tampering and third-party debugging.

For more information on how Dotfuscator fits into a secure software development lifecycle, see PreEmptive Solutions' [SDL App Protection page](#).

About Dotfuscator CE

Your copy of Microsoft Visual Studio 2017 includes a copy of **PreEmptive Protection - Dotfuscator Community Edition**, also known as Dotfuscator CE, free for personal use. For instructions on how to install the version of Dotfuscator CE included with Visual Studio 2017, see the [Installation page](#).

Dotfuscator CE offers a range of **software protection and hardening** services for developers, architects and testers. Examples of **.NET Obfuscation** and other **Application Protection** features included in Dotfuscator CE are:

- **Renaming** of identifiers to make reverse-engineering of the compiled assemblies more difficult.
- **Anti-tamper** to detect the execution of tampered applications, transmit incident alerts, and terminate tampered sessions.
- **Anti-debug** to detect the attachment of a debugger to a running application, transmit incident alerts, and terminate debugged sessions.
- **Application expiration behaviors** that encode an "end-of-life" date, transmit alerts when applications are

executed after their expiration date, and terminate expired application sessions.

- [Exception tracking](#) to monitor unhandled exceptions occurring within the application.
- [Session and feature usage tracking](#) to determine what applications have been executed, what versions of those applications, and what features are used in those applications.

For details on these features, including how they fit into your application protection strategy, see the [Capabilities page](#).

Dotfuscator CE offers basic protection out-of-the-box. Even more application protection measures are available to registered users of Dotfuscator CE, and to users of *PreEmptive Protection - Dotfuscator Professional Edition*, the world's leading [.NET Obfuscator](#). For information about enhancing Dotfuscator, see the [Upgrades page](#).

Getting Started

To begin using Dotfuscator CE from Visual Studio, type `dotfuscator` into the **Quick Launch** (Ctrl+Q) search bar.

- If Dotfuscator CE is already installed, this will bring up the *Menu* option to start the Dotfuscator CE user interface. For details, see [the Getting Started page of the full Dotfuscator CE User Guide](#).
- If Dotfuscator CE is not yet installed, this will bring up the relevant *Install* option. See the [Installation page](#) for details.

You can also get the **latest version** of Dotfuscator CE from [the Dotfuscator Downloads page on preemptive.com](#).

Full Documentation

This page and its sub-pages provide a high-level overview of Dotfuscator CE's features, as well as [instructions for installing the tool](#).

Please see [the full Dotfuscator CE User Guide at preemptive.com](#) for detailed usage instructions, including [how to start using the Dotfuscator CE user interface](#).

Capabilities of Dotfuscator

10/31/2017 • 2 min to read • [Edit Online](#)

This page focuses on the capabilities of Dotfuscator Community Edition (Dotfuscator CE) with some references to advanced options available through [upgrades](#).

Dotfuscator is a *post-build* system for .NET applications. With Dotfuscator CE, Visual Studio users are able to [obfuscate assemblies](#) and inject [active defense](#) and [analytics tracking](#) into the application - all without Dotfuscator needing to access the original source code. Dotfuscator protects your application in multiple ways, creating a layered protection strategy.

Dotfuscator CE supports a wide range of .NET assembly and application types, including [Universal Windows Platform \(UWP\)](#) and [Xamarin](#).

Intellectual Property Protection

Your application's design, behavior, and implementation are forms of intellectual property (IP). However, applications created for the .NET Framework are essentially open books; it's very easy to reverse engineer .NET assemblies, [as they contain high-level metadata and intermediate code](#).

Dotfuscator CE includes basic [.NET obfuscation](#) in the form of [renaming](#). Obfuscating your code with Dotfuscator reduces the risk of unauthorized access to source code through reverse engineering, as important naming information will no longer be public. Obfuscation also shows effort on your part to protect your code from examination - a valuable step in establishing that your IP is legally protected as trade secret.

Many of the [application integrity protection features](#) of Dotfuscator CE further hinder reverse engineering. For instance, a bad actor may attempt to attach a debugger to a running instance of your application in order to understand the program logic. Dotfuscator can inject [anti-debug behavior](#) into your application to obstruct this.

Application Integrity Protection

In addition to protecting your source code, it's also important to ensure your application is used as designed. Attackers can attempt to hijack your application in order to circumvent licensing policies (i.e., software piracy), to steal or manipulate sensitive data handled by the application, or to change the behavior of the application.

Dotfuscator CE can inject [application validation code](#) into your assemblies, including [anti-tamper](#) and [anti-debug](#) measures. When an invalid application state is detected, the validation code can [call upon application code to address to the situation in an appropriate way](#). Or, if you prefer not to write code to handle invalid uses of the application, Dotfuscator can also inject [telemetry reporting](#) and [response](#) behaviors, without requiring any modification to your source code.

Many of these same methods may also be used to enforce [end-of-life deadlines](#) for evaluation or trial software.

Application Monitoring

When developing an application, it is critical to understand the behavior patterns of users, including beta testers and users of prior versions. Application analytics allows you to track how frequently the application is used and how it is used, including what errors customers experience.

Dotfuscator CE can inject [exception-tracking](#), [session-tracking](#), and [feature-tracking](#) code into your application. When run, the processed application will transmit analytics data to a configured [PreEmptive Analytics endpoint](#).

See Also

[This topic in the full Dotfuscator CE User Guide](#)

Install Dotfuscator Community Edition (CE)

10/31/2017 • 1 min to read • [Edit Online](#)

Visual Studio 2017 introduces a new low-impact installation experience. As a result, Dotfuscator Community Edition (Dotfuscator CE) is not installed by default. However, it is easy to install Dotfuscator CE even if you have already installed Visual Studio.

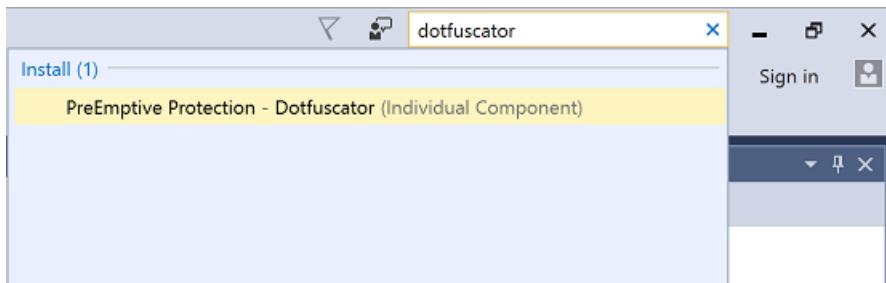
NOTE

In addition to the versions of Dotfuscator CE shipped with releases of Visual Studio, PreEmptive Solutions also periodically provides updated versions on its website. If you want to download the **latest version** directly instead of installing from Visual Studio, [click here to go to the Dotfuscator Downloads page](#).

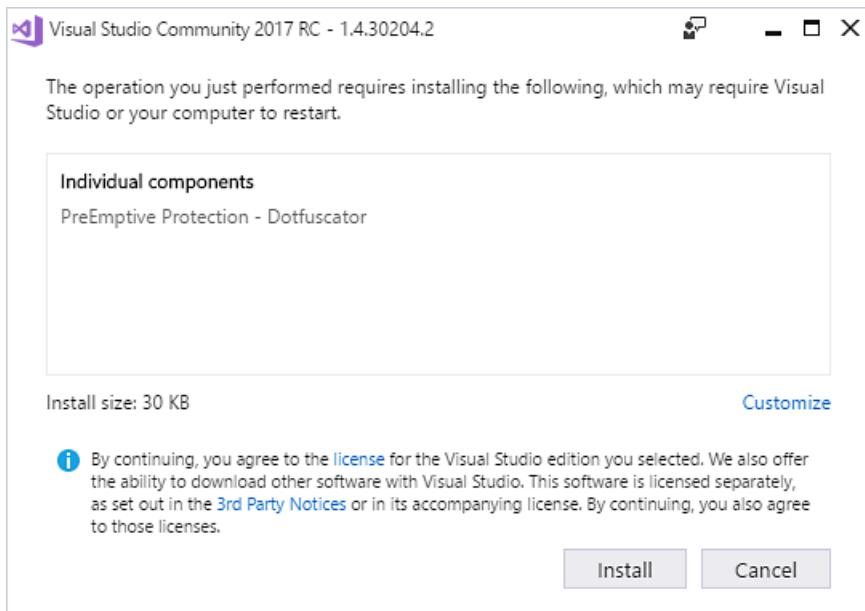
Within Visual Studio

You can install Dotfuscator CE from the Visual Studio IDE:

1. In the **Quick Launch** (Ctrl+Q) search bar, type `dotfuscator`.



2. In the Quick Launch results shown, under the *Install* heading, select **PreEmptive Protection - Dotfuscator (Individual Component)**.
 - If you instead see, under the *Menus* heading, **Tools - PreEmptive Protection - Dotfuscator**, then Dotfuscator CE is already installed. For usage details, see [the Getting Started page of the full Dotfuscator CE User Guide](#).
3. A Visual Studio Installer window will launch, pre-configured with to install Dotfuscator CE.
 - You may be required to provide administrator credentials to continue.
4. Close all instances of the Visual Studio IDE.

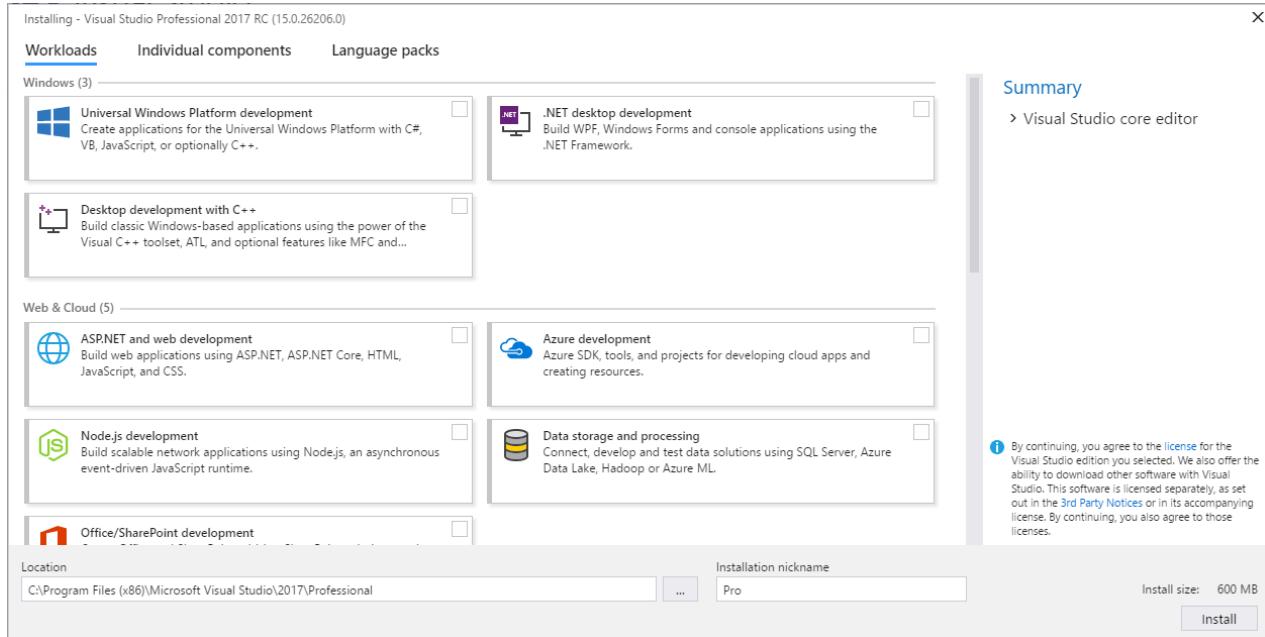


5. In the Visual Studio Installer window, click *Install*.

Once the installation is complete, you can start using Dotfuscator CE. For details, see [the Getting Started page of the full Dotfuscator CE User Guide](#).

During Visual Studio Installation

If you have not yet installed Visual Studio 2017, you can obtain the installer from [the Visual Studio website](#). When run, it will display installation options for the selected Visual Studio edition.

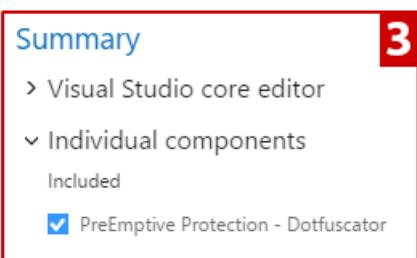


You can then install Dotfuscator CE as an individual component of Visual Studio 2017:

1. Select the **Individual components** tab.
2. Under *Code tools*, check the *PreEmptive Protection - Dotfuscator* item.



3. The *Summary* panel displays *PreEmptive Protection - Dotfuscator* under the *Individual Components* section.



4. Configure any further installation settings as appropriate for your environment.

5. When ready to install Visual Studio, click the *Install* button.

Once the installation is complete, you can start using Dotfuscator CE. For details, see [the Getting Started page of the full Dotfuscator CE User Guide](#).

See Also

[This topic in the full Dotfuscator CE User Guide](#)

Upgrade Dotfuscator Community Edition (CE)

10/31/2017 • 2 min to read • [Edit Online](#)

Dotfuscator Community Edition (Dotfuscator CE) offers many application protection and hardening features immediately to all developers using Microsoft Visual Studio. However, there are more features available to users who upgrade their version of Dotfuscator.

Registering Dotfuscator CE

Registered users of Dotfuscator CE get access to additional features, such as [command line support](#), which makes it easy to integrate Dotfuscator CE into your automated build process. In addition, registering will grant access to Lucidator, a built-in tool used for [decoding obfuscated stack traces](#).

Registration is quick, simple, and free of charge. To register Dotfuscator CE, see [the Registering Dotfuscator CE section on the Getting Started page of the full Dotfuscator CE User Guide](#).

Dotfuscator Professional

While Dotfuscator Community Edition provides a basic level of protection, **PreEmptive Protection - Dotfuscator Professional Edition** includes enhanced obfuscation transforms and protection capabilities. These include:

- *Intellectual Property Protection*
 - Additional renaming options, including Enhanced Overload Induction™ and randomized identifier selection.
 - Tooling for decoding obfuscated stack traces.
 - Access to enterprise-level obfuscation transforms, including [transforms targeted at defeating automated code decompilation](#).
 - The ability to [obscure sensitive strings](#), making a simple search of the decompiled code impossible.
 - The ability to [discreetly embed ownership and distribution strings into your assemblies](#) (software watermarking), allowing you to determine the source of unauthorized software leaks.
 - The ability to [combine multiple assemblies into one](#), making it even more difficult for attackers to determine the roles of code elements, as separation of concerns has been eliminated.
 - The ability to [automatically remove unused code from your application](#), reducing the amount of sensitive code that is shipped.
- *Application Integrity Protection*
 - Additional [application defense behaviors](#).
 - The ability to provide a warning period before an application's end-of-life deadline.
 - The ability to notify application code during an end-of-life warning period or after the deadline.
 - Telemetry encryption.
- *Application Monitoring*
 - The ability to collect and save collected information during temporary network outages.
 - The ability to collect personally-identifiable information.
 - Unlimited use of [feature tracking](#).
 - The ability to track exceptions caught and thrown by your code, in addition to unhandled exceptions.
 - The ability to track exceptions in `.dll` assemblies.
 - Telemetry encryption.

Dotfuscator Professional is the industry standard [.NET Obfuscator](#) and is suitable for enterprise developers

requiring ongoing support, maintenance, and product updates. Additionally, Dotfuscator Professional offers tighter integration with Visual Studio and is licensed for commercial use.

For more information on the advanced application protection features of Dotfuscator Professional, please visit PreEmptive Solutions' [Dotfuscator Overview page](#) and [compare it to Community Edition](#). [Fully-supported trials are available on request at preemptive.com](#).

See Also

[This topic in the full Dotfuscator CE User Guide](#)

What's New in Visual Studio 2017

3/15/2018 • 19 min to read • [Edit Online](#)

Updated for the **15.6 release**

Looking to upgrade from a previous version of Visual Studio? Here's what Visual Studio 2017 can offer you: Unparalleled productivity for any dev, any app, and any platform. Use Visual Studio 2017 to develop apps for Android, iOS, Windows, Linux, web, and cloud. Code fast, debug and diagnose with ease, test often, and release with confidence. You can also extend and customize Visual Studio by building your own extensions. Use version control, be agile, and collaborate efficiently with this release!

Here's a high-level recap of the changes we've made since our previous version, Visual Studio 2015:

- **Redefined fundamentals.** A new setup experience means that you can install more quickly and install what you want when you need it. Whether you want to load large solutions and projects, or work on folders of code, or even a single file of code, Visual Studio starts faster. And, Visual Studio helps you stay focused on the big picture, especially for teams embracing DevOps.
- **Performance and productivity.** We have focused on new and modern mobile, cloud, and desktop development capabilities. And, we've also improved the overall acquisition, performance, and general developer productivity experiences. Visual Studio starts faster, is more responsive, and uses less memory than before.
- **Cloud app development with Azure.** A built-in suite of Azure tools enable you to easily create cloud-first apps powered by Microsoft Azure. Visual Studio makes it easy to configure, build, debug, package, and deploy apps and services on Azure.
- **Windows app development.** Use the UWP templates in Visual Studio 2017 to create a single project for all Windows 10 devices – PC, tablet, phone, Xbox, HoloLens, Surface Hub, and more. Then, produce an app package and submit it to Microsoft Store from within Visual Studio to get your app out to customers.
- **Mobile app development.** In Visual Studio 2017, you can innovate and get results fast with Xamarin, which unifies your multi-platform mobile requirements by using one core codebase and set of skills. Go mobile with your existing teams, technology investments, and C# code to deliver consumer-grade experiences ahead of schedule and under budget. Accelerate every step of the mobile lifecycle to deliver world-class consumer experiences or a portfolio of productivity apps to empower your workforce.
- **Cross-platform development.** Seamlessly deliver software to any targeted platform. Extend DevOps processes to SQL Server through Redgate Data Tools and safely automate database deployments from Visual Studio. Or, use .NET Core to write apps and libraries that run unmodified across Windows, Linux, and macOS operating systems. (And **new in 15.3**: Get side-by-side support for .NET Core 2.0 SDKs.)
- **Games development.** With Visual Studio Tools for Unity (VSTU), you can use Visual Studio to write game and editor scripts in C# and then use its powerful debugger to find and fix errors. The latest release of VSTU includes syntax coloring for Unity's ShaderLab shader language, better debugger visualizations, and improved code generation for the MonoBehavior wizard. VSTU also brings your Unity project files, console messages, and the ability to start your game into Visual Studio so you can spend less time switching to and from the Unity Editor while writing code.
- **AI development.** With Visual Studio Tools for AI (**new in 15.5**), you can use the productivity features of Visual Studio to accelerate AI innovation. Build, test, and deploy Deep Learning / AI solutions that seamlessly integrate with Azure Machine Learning for robust experimentation capabilities, such as submitting data preparation and model training jobs transparently to different compute targets. And, Visual Studio Tools for AI provides support for custom metrics and run history tracking, which enable data science reproducibility and auditing.

NOTE

For a complete list of new features and functionality in Visual Studio 2017, see the [current Release Notes](#). And for a peek at future feature offerings, see the [Preview Release Notes](#).

Here's more detailed information about some of the most notable improvements and new features in Visual Studio 2017.

Redefined fundamentals

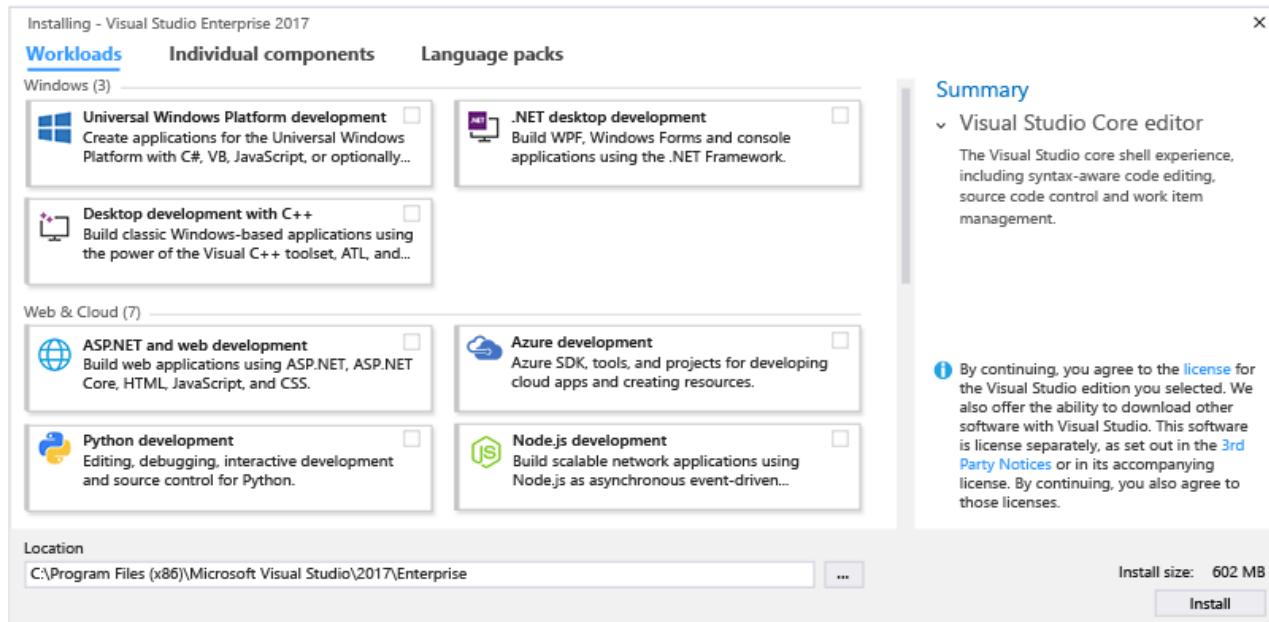
A new setup experience

[Download Visual Studio 2017](#) or [Check Visual Studio system requirements](#)

Visual Studio makes it easier and faster to install just the features you need, when you need them. And, it uninstalls cleanly, too.

The most important change to note when you install Visual Studio is its new setup experience. On the **Workloads** tab, you'll see installation options that are grouped to represent common frameworks, languages, and platforms. It covers everything from .NET desktop development to C++ application development on Windows, Linux, and iOS.

Choose the workloads you need, and change them when you need to.



Want to pick your own components instead of using workloads? Select the **Individual components** tab from the installer. Want to install Language Packs without also having to change the Windows language option? Choose the **Language packs** tab of the installer.

To learn more about the new installation experience, including step-by-step instructions that walk you through it, see the [Install Visual Studio](#) page.

A focus on accessibility

New in 15.3, we made over 1,700 targeted fixes to improve compatibility between Visual Studio and the assistive technologies that many customers use. There are dozens of scenarios that are more compatible with screen readers, high contrast themes, and other assistive technologies than ever before. The debugger, editor and shell have all gotten significant improvements, too.

For more information, see the [Accessibility improvements in Visual Studio 2017 version 15.3](#) blog post.

Performance and productivity

Sign in across multiple accounts

We've introduced a new identity service in Visual Studio that allows you to share user accounts across Team Explorer, Azure Tools, Microsoft Store publishing, and more.

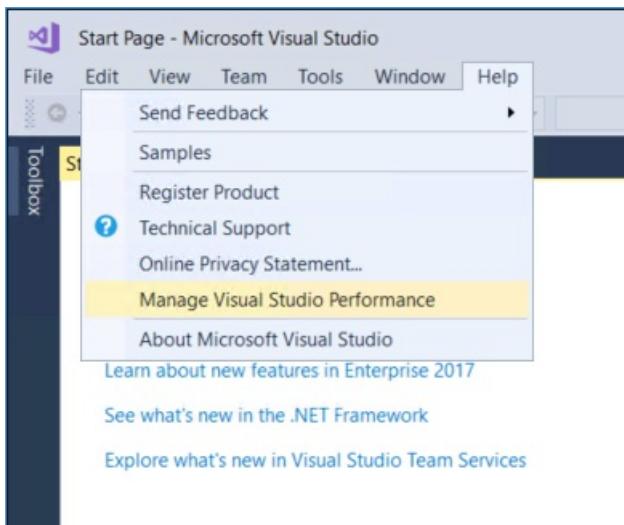
You can stay signed in longer, too. Visual Studio won't ask you to sign in again every 12 hours. To learn more, see the [Fewer Visual Studio Sign-in Prompts](#) blog post.

Start Visual Studio faster

The new Visual Studio Performance Center can help you optimize your IDE start-up time. The Performance Center lists all the extensions and tool windows that might slow down the IDE startup. You can use it to improve startup performance by determining when extensions start, or whether tool windows are open at startup.

Faster on-demand loading of extensions

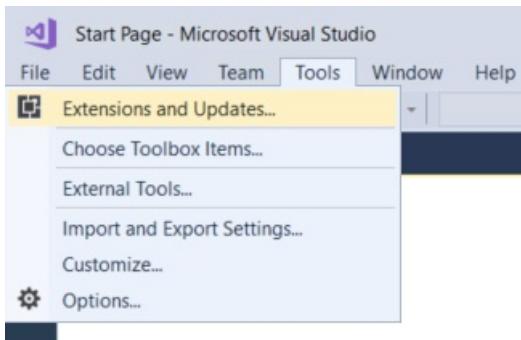
Visual Studio is moving its extensions (and working with third-party extensions too) so that they load on-demand, rather than at IDE startup. Curious about which extensions impact startup, solution load, and typing performance? You can see this information in Help -> Manage Visual Studio Performance.



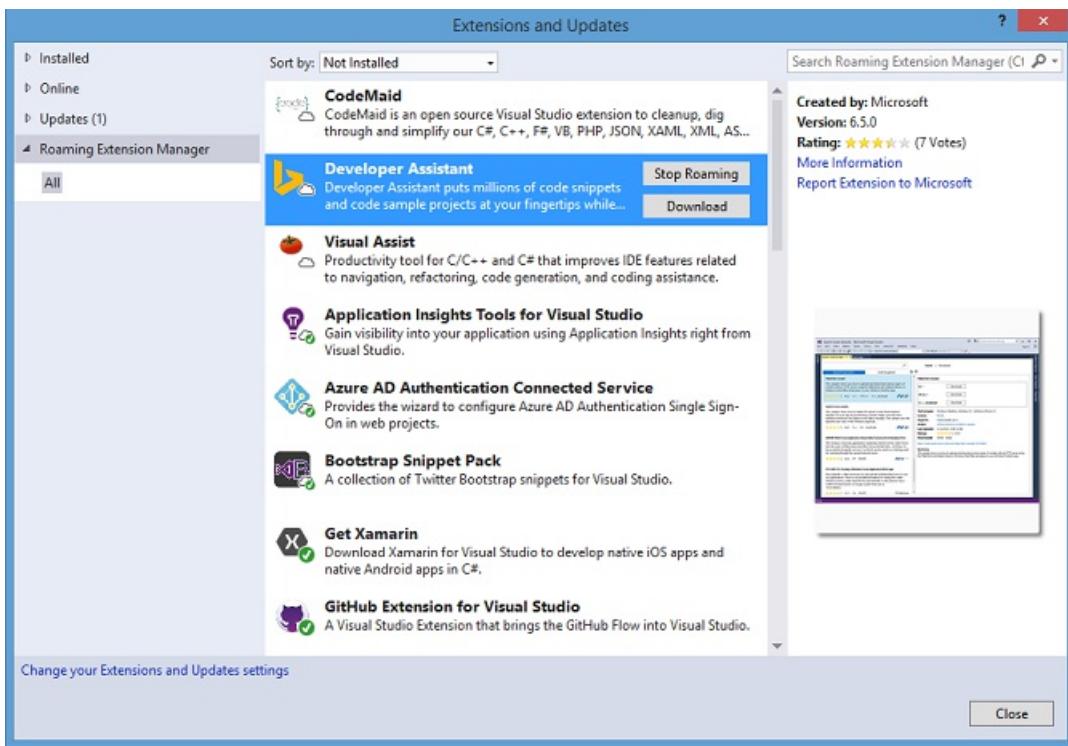
Manage your extensions with Roaming Extensions Manager

It's easier to set up each development environment with your favorite extensions when you sign in to Visual Studio. The new Roaming Extension Manager keeps track of all your favorite extensions by creating a synchronized list in the cloud.

To see a list of your extensions in Visual Studio, click Tools > Extensions & Updates, and then click the Roaming Extension Manager.



The Roaming Extension Manager tracks all the extensions you install, but you can choose which ones you want to add to your Roaming list.



When you use the Roaming Extension Manager, there are three icon types on your list:

- **Roamed**: An extension that is part of this Roaming List, but not installed on your machine. (You can install these by using the **Download** button.)
- **Roamed & Installed**: All extensions that are part of this Roaming List and installed in your dev environment. (If you decide you do not want to roam, you can remove these by using the **Stop Roaming** button.)
- **Installed**: All extensions that are installed in this environment, but are not part of your Roaming List. (You can add extensions to the Roaming List by using the **Start Roaming** button.)

Any extension that you download while you are signed in is added to your list as **Roamed & Installed** and is part of your Roaming list, which gives you access to it from any machine.

Experience live unit testing

In Visual Studio Enterprise 2017, live unit testing gives you live unit test results and code coverage in the editor while you are coding. It works with C# and Visual Basic projects for both the .NET Framework and .NET Core, and it supports three test frameworks of MSTest, xUnit, and NUnit.

The screenshot shows a Visual Studio code editor with two tabs: 'UnitTest1.cs' and 'Class1.cs'. The 'Class1.cs' tab is active, displaying C# code for a utility library. The code defines a static class 'StringLibrary' with three methods: 'StartsWithUpper', 'StartsWithLower', and 'GetWordCount'. Each method has its own test coverage summary. The 'StartsWithUpper' method has 3 references, 3 passing, and 0 failing. The 'StartsWithLower' method has 3 references, 2 passing, and 1 failing. The 'GetWordCount' method has 0 references. The code uses 'using System;' and 'using System.Text.RegularExpressions;'. The 'String' class from 'System' is imported.

```
1  using System;
2  using System.Text.RegularExpressions;
3
4  namespace UtilityLibraries
5  {
6      public static class StringLibrary
7      {
8          // 2 references
9          // 3 references | 3/3 passing
10         public static bool StartsWithUpper(this String str)
11         {
12             if (String.IsNullOrEmpty(str))
13                 return false;
14
15             Char ch = str[0];
16             return Char.IsUpper(ch);
17         }
18
19         // 3 references | 2/3 passing
20         public static bool StartsWithLower(this String str)
21         {
22             if (String.IsNullOrEmpty(str))
23                 return false;
24
25             Char ch = str[0];
26             return Char.IsLower(ch);
27         }
28
29         // 0 references
30         public static int GetWordCount(this String str)
31         {
32             return Regex.Matches(str, @"\w+").Count;
33         }
34     }
35 }
```

For more information, see the [Introducing Live Unit Testing](#). For a list of new features added in each release of Visual Studio Enterprise 2017, see [What's new in Live Unit Testing](#).

Setting up a CI/CD pipeline

Automated testing

Automated testing is a key part of any DevOps pipeline. It allows you to consistently and reliably test and release your solution on much shorter cycles. CI/CD (Continuous Integration and Continuous Delivery) flows can help make the process more efficient.

For more information about automated tests, see the [CI/CD pipeline for automated tests in DevOps](#) blog post.

And, for more information about what's new in the [Continuous Delivery Tools for Visual Studio](#) DevLabs extension, see the [Committing with Confidence: Commit Time Code Quality](#) blog post.

Visual Studio IDE enhancements

Use new refactorings

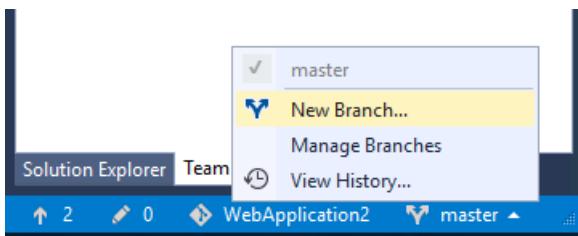
Refactoring is the process of improving your code after it has been written. Refactoring changes the internal structure of the code without changing its behavior. We add new refactorings often; here are just a few:

- Add parameter (from CallSite)
- Generate overrides
- Add named argument
- Add null-check for parameters
- Insert digit-separators into literals
- Change base for numeric literals (for example, hex to binary)
- Convert if-to-switch
- Remove unused variable

For more information, see [Quick Actions](#).

Interact with Git

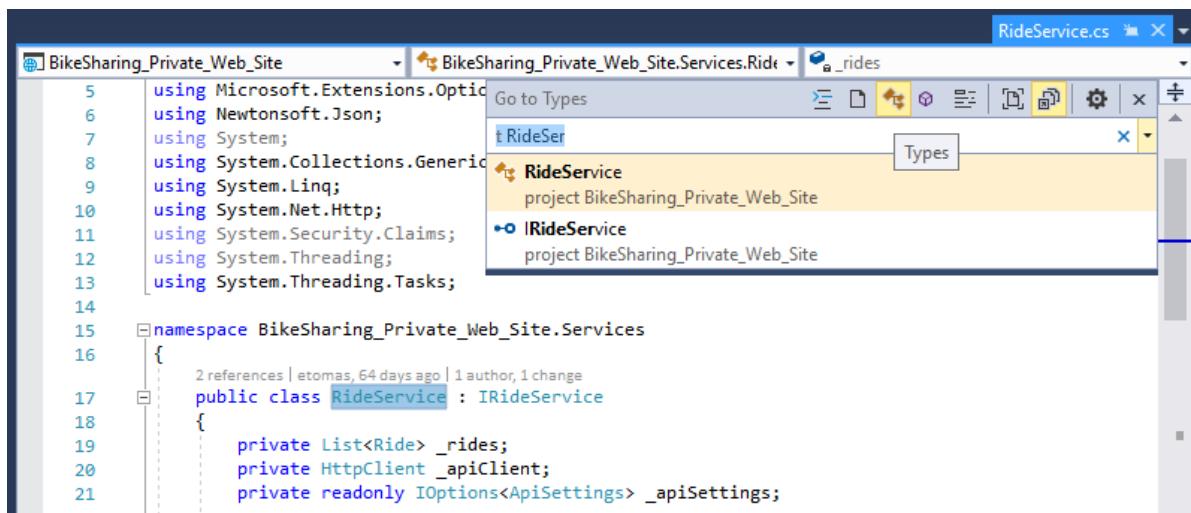
When you are working with a project in Visual Studio, you can set up and quickly commit and publish your code to a Git service. You can also manage your Git repositories by using menu clicks from buttons in the bottom right-hand corner of the IDE.



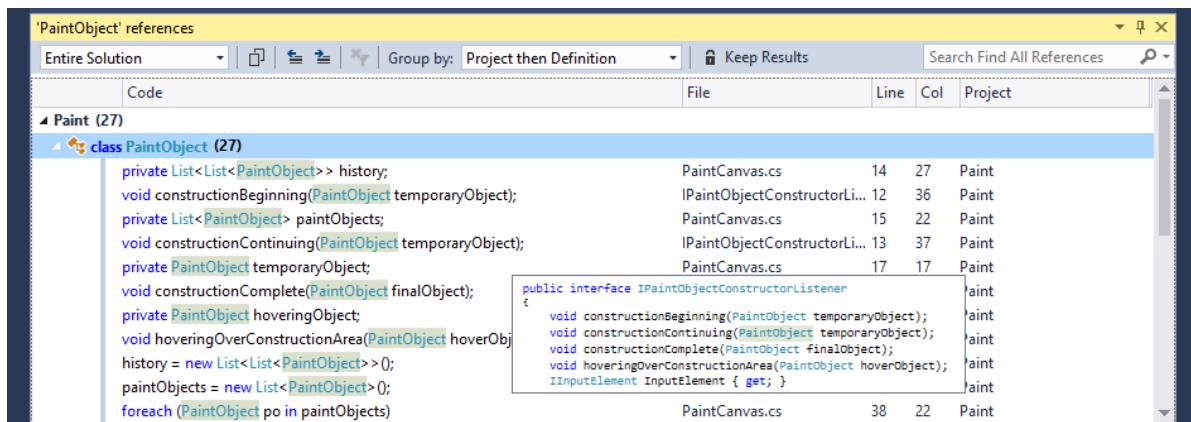
Experience improved navigation controls

We've refreshed the navigation experience to help you get from A to B with greater confidence and fewer distractions.

- **New in 15.4: Go To Definition (Ctrl+click or F12)** – Mouse users have an easier way to navigate to the definition of a member by pressing **Ctrl** and then clicking the member. Pressing **Ctrl** and hovering over a code symbol will underline it and turn it into a link. See [Go To Definition](#) and [Peek Definition](#) for more information.
- **Go To Implementation (Ctrl+F12)** – Navigate from any base type or member to its various implementations.
- **Go To All (Ctrl+T or Ctrl+,)** – Navigate directly to any file/type/member/symbol declaration. You can filter your result list or use the query syntax (for example, "f searchTerm" for files, "t searchTerm" for types, etc.).



- **Find All References (Shift+F12)** – With syntax colorization, you can group Find All Reference results by a combination of project, definition, and path. You can also "lock" results so that you can continue to find other references without losing your original results.



- **Structure Visualizer** – Dotted, gray vertical lines (indent guides) act as landmarks in code to provide context within your frame of view. You may recognize them from the popular Productivity Power Tools. You can use them to visualize and discover what block of code you're in at any time without having to scroll. Hovering over the lines displays a tooltip that shows you the opening of that block and its parents. It's available for all the languages supported via TextMate grammars, as well as C#, Visual Basic, and XAML.

```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello world");
        }
    }
}
```

For more information about the new productivity features, see the [Productivity in Visual Studio 2017](#) blog post by Mark Wilson-Thomas.

Visual C++

You'll see several improvements in Visual Studio, such as distributing C++ Core Guidelines with Visual Studio, updating the compiler by adding enhanced support for C++11 and C++ features, and adding and updating functionality in the C++ libraries. We've also improved the performance of the C++ IDE, installation workloads, and more.

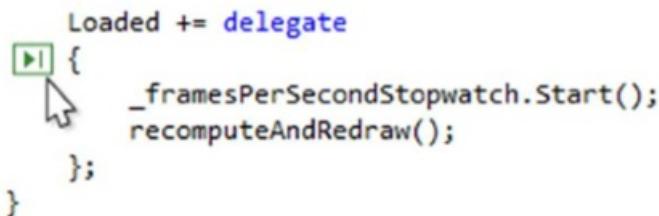
As well, we've fixed over 250 bugs and reported issues in the compiler and tools, many submitted by customers through [Microsoft Connect](#).

For complete details, see the [What's New for Visual C++ in Visual 2017](#) page.

Debugging and Diagnostics

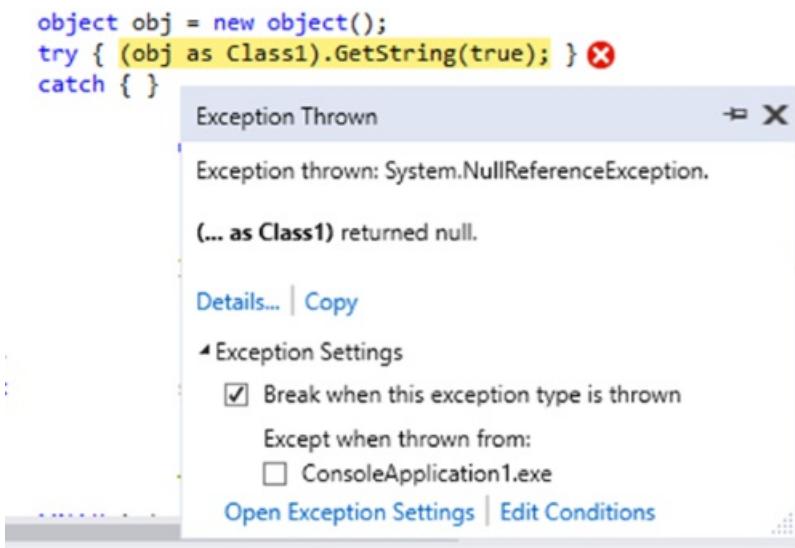
Run to Click:

Now, you can more easily skip ahead during debugging without setting a breakpoint to stop on the line you want. When you are stopped in the debugger, just click the icon that appears next to the line of code. Your code will run and stop on that line the next time it is hit in your code path.



The new Exception Helper:

The new Exception Helper helps you view your exception information at-a-glance. The information is presented in a compact form with instant access to inner exceptions. When you diagnose a NullReferenceException, you can quickly see what was null right inside the Exception Helper.

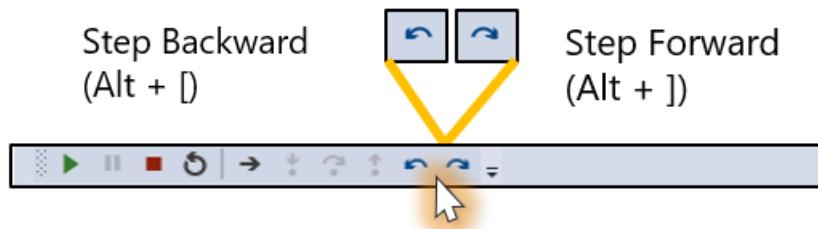


For more information, see the [Using the New Exception Helper in Visual Studio](#) blog post.

Snapshots and IntelliTrace step-back:

New in 15.5: IntelliTrace step-back automatically takes a snapshot of your application at every breakpoint and debugger step event. The recorded snapshots enable you to go back to previous breakpoints or steps and view the state of the application as it was in the past. IntelliTrace step-back can save you time when you want to see the previous application state but don't want to restart debugging or recreate the desired app state.

You can navigate and view snapshots by using the **Step Backward** and **Step Forward** buttons in the Debug toolbar. These buttons navigate the events that appear in the **Events** tab in the **Diagnostic Tools** window. Stepping backward or forward to an event automatically activates historical debugging on the selected event.



For more information, see the [View snapshots using IntelliTrace step-back](#) page.

Containerization

Containers provide you with increased app density and lower deployment cost along with improved productivity and DevOps agility.

Docker Container Tooling

New in 15.5

- Visual Studio includes tools for Docker containers that now support multi-stage Dockerfiles, which streamline creating optimized container images.
- By default, Visual Studio will automatically pull, build, and run the necessary container images in the background when you open a project that has Docker support. You can disable this via the **Automatically start containers in background** setting in Visual Studio.

Cloud app development with Azure

Azure Functions Tools

As part of the "Azure development" workload, we've included tools to help you develop Azure functions by using

pre-compiled C# class libraries. Now you can build, run, and debug on your local development machine and then publish directly to Azure from Visual Studio.

For more information, see the [Azure Functions Tools for Visual Studio](#) page.

Debug live ASP.NET apps using snappoints and logpoints in live Azure applications

New in 15.5: The Snapshot Debugger takes a snapshot of your in-production apps when code that you are interested in executes. To instruct the debugger to take a snapshot, you set snappoints and logpoints in your code. The debugger lets you see exactly what went wrong, without impacting traffic of your production application. The Snapshot Debugger can help you dramatically reduce the time it takes to resolve issues that occur in production environments.

Snapshot collection is available for the following web apps running in Azure App Service:

- ASP.NET applications running on .NET Framework 4.6.1 or later.
- ASP.NET Core applications running on .NET Core 2.0 or later on Windows.

For more information, see [Debug live ASP.NET apps using snappoints and logpoints](#).

Windows app development

Universal Windows Platform

The Universal Windows Platform (UWP) is the app platform for Windows 10. You can develop apps for UWP with just one API set, one app package, and one store to reach all Windows 10 devices – PC, tablet, phone, Xbox, HoloLens, Surface Hub, and more. UWP supports different screen sizes and a variety of interaction models, whether it be touch, mouse and keyboard, a game controller, or a pen. At the core of UWP apps is the idea that users want their experiences to be mobile across ALL their devices, and they want to use whatever device is most convenient or productive for the task at hand.



Choose your preferred development language—from C#, Visual Basic, C++, or JavaScript—to create a Universal Windows Platform app for Windows 10 devices. Visual Studio 2017 provides a UWP app template for each language that lets you create a single project for all devices. When your work is finished, you can produce an app package and submit it to Microsoft Store from within Visual Studio to get your app out to customers on any Windows 10 device.

New in 15.5

Visual Studio 2017 version 15.5 provides the best support for the Windows 10 Fall Creators Update SDK (10.0.16299.0). The Windows 10 Fall Creators Update also brings many improvements for UWP developers. Here are some of the biggest changes:

- **Support for .NET Standard 2.0**

In addition to streamlined app deployment, the Windows 10 Fall Creators Update is the first release of

Windows 10 to provide .NET Standard 2.0 support. Effectively, [.NET Standard](#) is a reference implementation of the base class library that any .NET platform can implement. The goal of .NET Standard is to make it as easy as possible for .NET developers to share code across any .NET platform they choose to work on.

- **The best of both UWP and Win32**

We have improved the Windows 10 Platform with the [Desktop Bridge](#) to make Windows 10 better for all .NET developers, whether their current focus is on UWP, WPF, Windows Forms, or Xamarin. With the new App Packaging project type in Visual Studio 2017 version 15.5, you can create Windows App Packages for your WPF or Windows Forms projects, just like you can for UWP projects. After you package your app, you get all the Windows 10 app deployment benefits and have the option to distribute via Microsoft Store (for consumer apps) or Microsoft Store for Business and Education. Because packaged apps have access to both the full UWP API surface and the Win32 APIs on desktop, you can now modernize your WPF and Windows Forms applications gradually with UWP APIs and Windows 10 features. Moreover, you can include your Win32 components in your UWP applications that light up on desktop with all Win32 capabilities.

For more information about UWP, see the [Develop apps for the Universal Windows Platform \(UWP\)](#) page.

Mobile app development

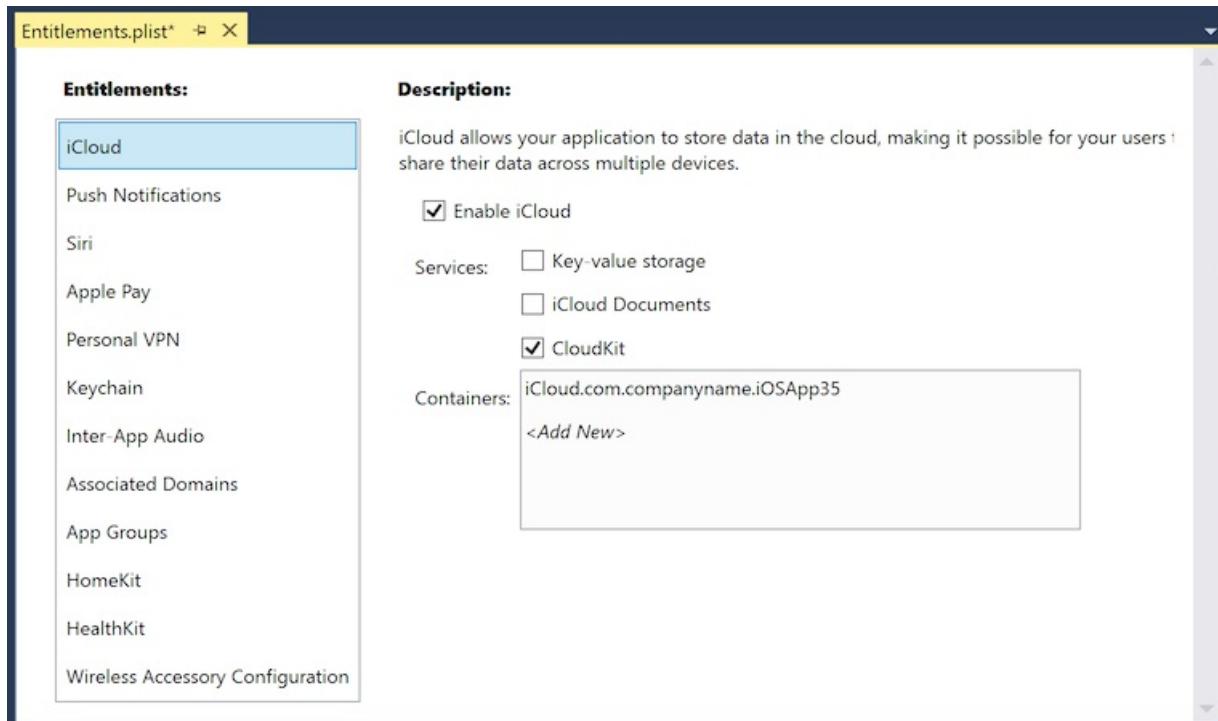
Xamarin

As part of the "Mobile development with .NET" workload, developers familiar with C#, .NET, and Visual Studio can deliver native Android, iOS, and Windows apps by using Xamarin. Developers can enjoy the same power and productivity when working with Xamarin for mobile apps, including remote debugging on Android, iOS, and Windows devices—without having to learn native coding languages like Objective-C or Java.

For more information, see the [Visual Studio and Xamarin](#) page.

Entitlements editor

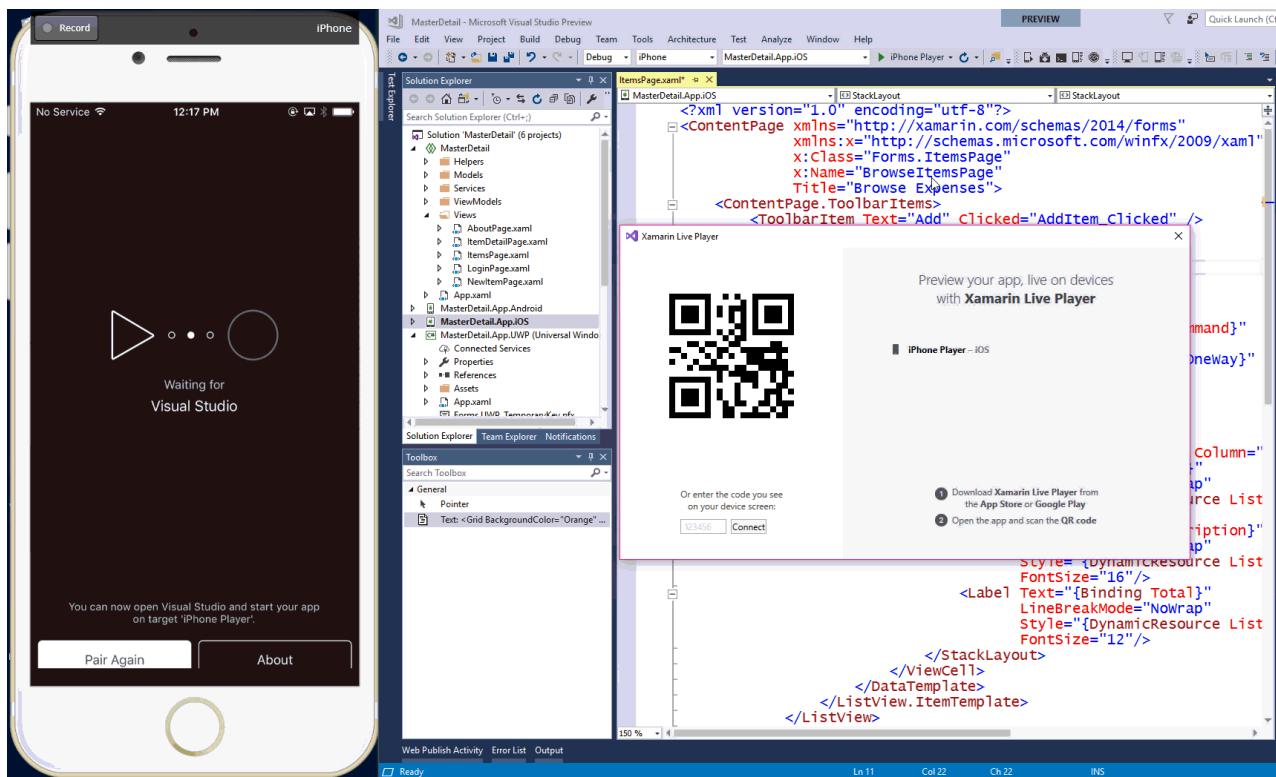
New in 15.3: For your iOS development needs, we've added a stand-alone Entitlements editor. It includes a user-friendly UI that can be easily browsed. To launch it, double-click your entitlements.plist file.



Visual Studio Tools for Xamarin

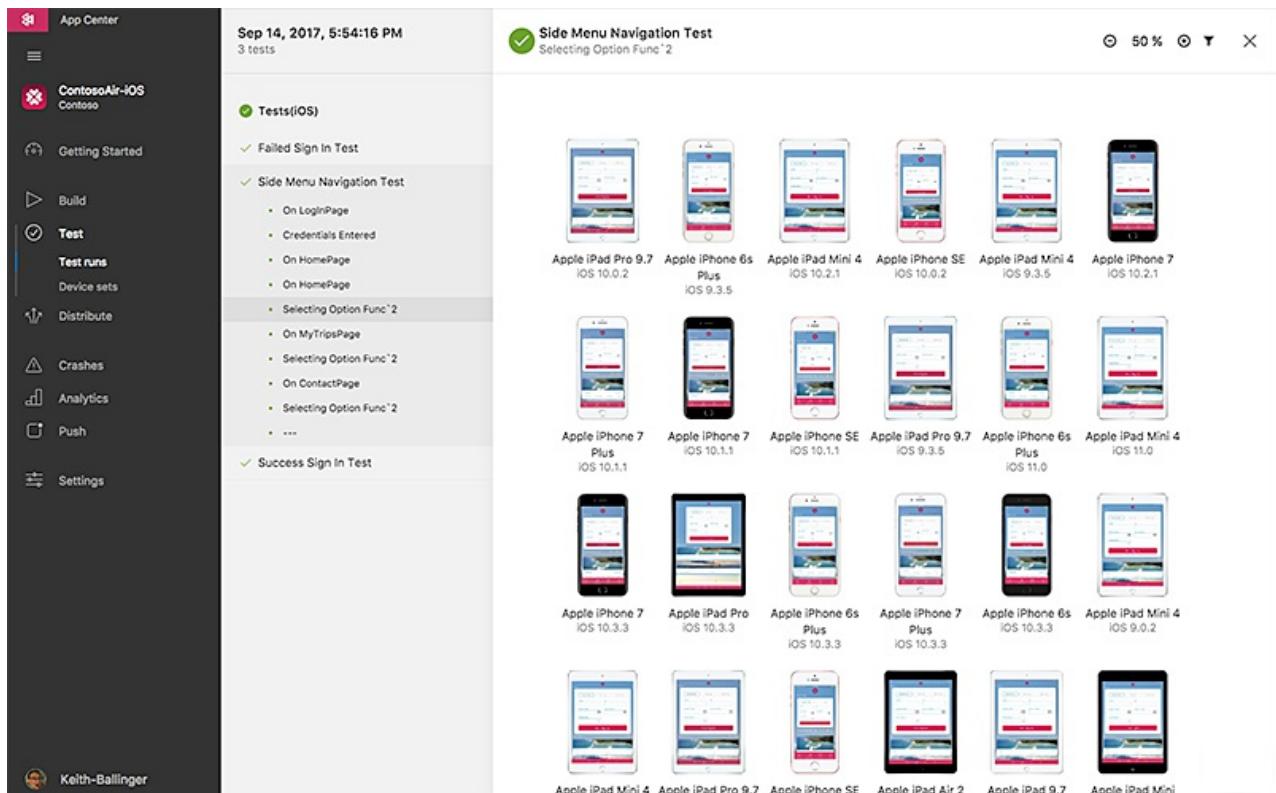
New in 15.4: Xamarin Live enables developers to continuously deploy, test, and debug their apps, directly on iOS and Android devices. After downloading the Xamarin Live Player—available in the App Store or on Google Play—you can pair your device with Visual Studio and revolutionize the way you build mobile apps. This functionality is

now included in Visual Studio and can be enabled by going to Tools > Options > Xamarin > Other > Enable Xamarin Live Player.



Visual Studio App Center

New in 15.5: Visual Studio App Center—which is now generally available for Android, iOS, macOS, and Windows apps—has everything you need to manage the lifecycle of your apps, including automated builds, testing on real devices in the cloud, distribution to beta testers and app stores, and monitoring of real-world usage through crash and analytics data. Apps written in Objective-C, Swift, Java, C#, Xamarin, and React Native are supported across all features.



For more information, see the [Introducing App Center: Build, Test, Distribute and Monitor Apps in the Cloud](#) blog post.

Cross-platform development

Redgate Data Tools

To extend DevOps capabilities to SQL Server database development, Redgate Data Tools are now available in Visual Studio.

Included with Visual Studio 2017 Enterprise:

- [Redgate ReadyRoll Core](#) helps you develop migration scripts, manage database changes using source control, and safely automate deployments of SQL Server database changes alongside applications changes.
- [Redgate SQL Prompt Core](#) helps you write SQL more quickly and accurately with the help of intelligent code completion. SQL Prompt autocompletes database and system objects and keywords, and offers column suggestions as you type. This results in cleaner code and fewer errors because you don't have to remember every column name or alias.

Included with all editions of Visual Studio 2017:

- [Redgate SQL Search](#) increases your productivity by helping you quickly find SQL fragments and objects across multiple databases.

To learn more, see the [Redgate Data Tools in Visual Studio 2017](#) blog post.

.NET Core

.NET Core is a general purpose, modular, cross-platform, and open source implementation of the .NET Standard and contains many of the same APIs as the .NET Framework.

The .NET Core platform is made of several components, which include the managed compilers, the runtime, the base class libraries, and numerous application models, such as ASP.NET Core. .NET Core supports three main operating systems: Windows, Linux, and macOS. You can use .NET Core in device, cloud, and embedded/IoT scenarios.

And, it now includes Docker support.

New in 15.3: Visual Studio 2017 version 15.3 supports .NET Core 2.0 development. Using .NET Core 2.0 requires downloading and installing the .NET Core 2.0 SDK separately.

For more information, see the [.NET Core Guide](#) page.

Games development

Visual Studio Tools for Unity

As part of the "Games development for Unity" workload, we've included tools to help you develop cross-platform to create 2D and 3D games and interactive content. Create once and publish to 21 platforms, including all mobile platforms, WebGL, Mac, PC and Linux desktop, web, or consoles by using Visual Studio 2017 and Unity 5.6.

For more information, see the [Visual Studio Tools for Unity](#) page.

AI development

Visual Studio Tools for AI

New in 15.5: Use the productivity features of Visual Studio to accelerate AI innovation today. Use built-in code editor features like syntax highlighting, IntelliSense, and text auto formatting. You can interactively test your deep learning application in your local environment by using step-through debugging on local variables and models.

```

# Create and train a feedforward classification model for MNIST images
def convnet_mnist(debug_output=False, epoch_size=60000, minibatch_size=64, max_epochs=10):
    image_height = 28
    image_width = 28
    num_channels = 1
    input_dim = image_height * image_width * num_channels
    num_output_classes = 10

    # Input variables denoting the feature and label data
    input_var = CNTK.ops.input_variable((num_channels, image_height, image_width), np.float32)
    label_var = CNTK.ops.input_variable((num_output_classes), np.float32)

    # Instantiate the feedforward classification model
    scaled_input = CNTK.ops.element_times(CNTK.ops.constant(0.00390625), input_var)

    with CNTK.layers.default_options(activation=CNTK.ops.relu, pad=False):
        conv1 = CNTK.layers.Convolution2D(5, 5, 32, pad=True)(scaled_input)
        pool1 = CNTK.layers.MaxPooling(conv1, stride=2, name="MaxPool1")
        conv2 = CNTK.layers.Convolution2D(5, 5, 64, pad=True)(pool1)
        pool2 = CNTK.layers.MaxPooling(conv2, stride=2, name="MaxPool2")
        conv3 = CNTK.layers.Convolution2D(5, 5, 128, pad=True)(pool2)
        f4 = CNTK.layers.Dense(128)(conv3)
        drop4 = CNTK.layers.Dropout(0.5)(f4)
        z = CNTK.layers.Dense(10)(drop4)

    ce = CNTK.losses.cross_entropy
    pe = CNTK.metrics.classification_error

    reader_train = create_reader()

    # Set learning parameters
    lr_per_sample = [0.001]*10
    lr_schedule = CNTK.learnin...
    mm_time_constant = [0]*5 + [10]
    mm_schedule = CNTK.learnin...

    # Instantiate the trainer object
    learner = CNTK.learners.momentum...
    progress_printers = [CNTK.log...
    progress_printers.append(TensorBoard(...))

    # Create the trainer
    trainer = CNTK.Trainer(z, ce, reader_train, learner, progress_printers)

```

For more information, see the [Visual Studio Tools for AI](#) page.

Talk to us

Why send feedback to the Visual Studio team? Because we take customer feedback seriously. It drives much of what we do.

If you want to make a suggestion about how we can improve Visual Studio, or report a problem, please see the [Talk to Us](#) page.

Report a problem

Sometimes, a message isn't enough to convey the full impact of a problem you've encountered. If you experience a hang, crash, or other performance issue, you can easily share repro steps and supporting files (such as screenshots, and trace and heap dump files) with us by using the **Report a Problem** tool. For more information about how to use this tool, see the [How to Report a Problem](#) page.

Track your issue in Connect

If want to track the status of your Visual Studio feedback, go to [Connect](#) and report the bug there. After you report it, you can return to Connect to track its status.

See Also

- [Visual Studio 2017 Release Notes](#)
- [What's New in Visual C++](#)
- [What's New in C#](#)
- [What's New for Team Foundation Server](#)
- [What's New in Visual Studio for Mac](#)

How to report a problem with Visual Studio 2017

3/12/2018 • 2 min to read • [Edit Online](#)

If you experience a problem with Visual Studio, we want to know about it. Here's how to report the problem so that we can diagnose and fix it.

Sign in to Visual Studio

If you haven't already done so, sign in to Visual Studio before you report a problem. That way, you can report a problem that you're experiencing, and also vote or comment on it. You can even vote or comment on any other problem that you see posted, too.

1. In Visual Studio, select **Help > Send Feedback > Report a Problem**.
2. If you are not signed in, select **Sign In**; it's on the right-hand side of the tool, as shown in the following screenshot.
3. Follow the instructions on-screen to sign in.



Search and vote for similar problems

1. Search for your problem and see if others have reported it, too.
2. If someone has reported it, "up-vote" it to let us know.

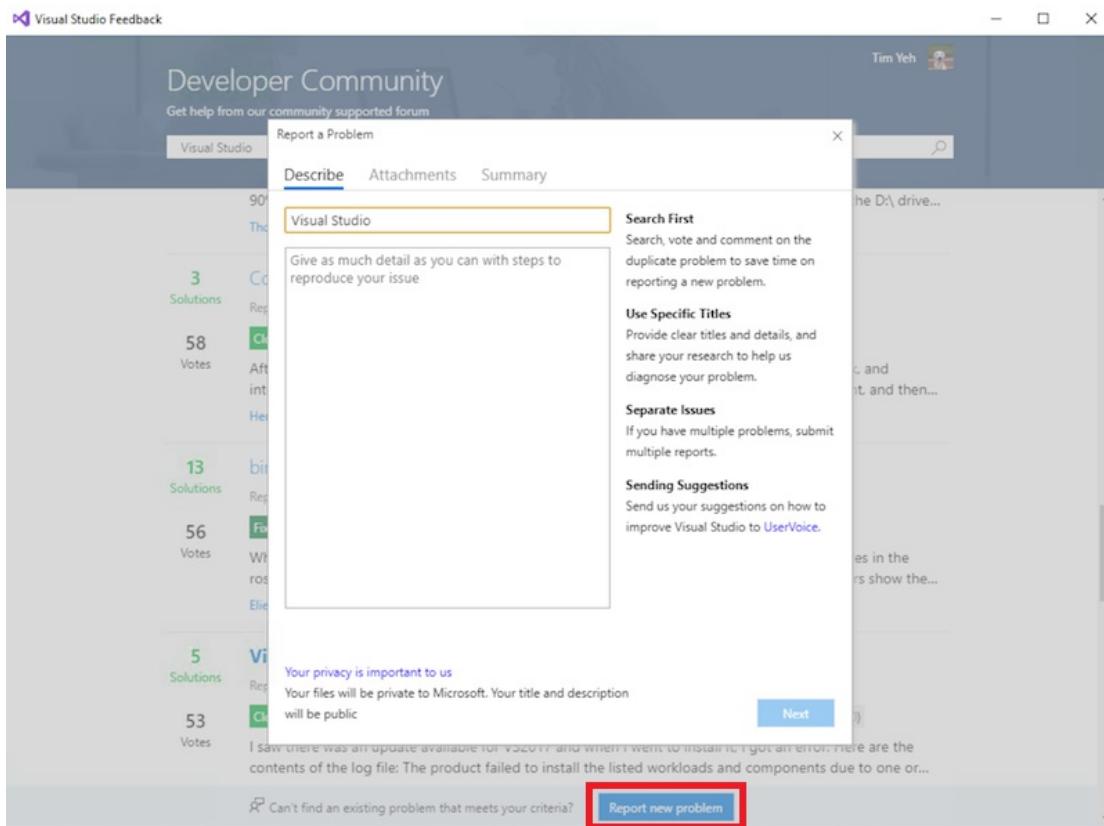
A screenshot of the Visual Studio Feedback application window titled "Visual Studio Feedback". The main area is labeled "Developer Community" with the sub-instruction "Get help from our community supported forum". Below this is a search bar with the placeholder "Visual Studio". The interface shows a list of reported problems:

- Solution appears to close immediately after opening**
4 Solutions
Reported by Andrew Jocelyn 3/7/2017, 5:59:06 PM
Closed - Fixed windows 10.0 visual studio 2017 ide fixed-in: Visual Studio 2017 RTW
Hi Visual Studio Pro 2017. I'm finding that when I open a solution it appears briefly in Solution Explorer then immediately closes with no signs of an error. Opening the solution again appears to work. The...
Praveen Potturu updated 10/31/2017
- Visual Studio may freeze or crash when running on a pen-enabled machine**
11 Solutions
Reported by Paul Chapman 5/11/2017, 8:47:06 PM
Fixed - Pending Release windows 10.0 visual studio 2017 visual studio 2017 installer (version 15.1) visual studio 2017 (version 15.2)
Under certain conditions, **Visual Studio** may terminate unexpectedly (a freeze / hang followed by a crash) when running on a pen-enabled machine with Windows 10 Creators Update, or other Windows versions...
Neeraj Tyagi [MSFT] updated 11/1/2017
- Debugger icons missing**
2 Solutions
Reported by Denis Osipov 8/18/2017, 1:42:37 PM
Closed - Fixed windows 10.0 visual studio 2017 version 15.3
No UI icons when browsing variables.

A "Report new problem" button is visible at the bottom right of the list.

Report a new problem

1. If you don't find what you're looking for, choose the **Report new problem** button at the bottom of the screen.
2. Create a descriptive title for the problem that helps us route it to the correct Visual Studio team.
3. Give us any additional details, and if possible, provide us with the steps to reproduce the problem.



Provide a screenshot and attachments (optional)

Choose to send your current screen to Microsoft. You can attach additional screenshots or other files by choosing the **Attach Additional Files** button.

Provide a trace and heap dump (optional)

Trace and heap dump files are useful in helping us diagnose problems. We appreciate it when you use the **Report a Problem** tool to record your repro steps and send the data to Microsoft. Here's how to do so.

1. Choose the **Record** tab.
2. Choose **Start Recording**. Give permission to run the tool.

[Describe](#) [Attachments](#) [Summary](#)[Screenshot](#) [Record](#) !

Record your actions to reproduce the issue.

Record your actions

| | |
|---------------------------------------|------------|
| Visual Studio Instance (window title) | Process ID |
| <Create a new instance> | |

[Start recording](#)**Attachments**[Attach Additional Files](#)

Your privacy is important to us

Your files will be private to Microsoft. Your title and description
will be public

[Previous](#)[Next](#)

3. When the **Steps Recorder** tool appears, perform the steps that reproduce the problem.
4. When you are done, choose the **Stop Record** button.
5. Wait a few minutes for Visual Studio to collect and package the information that you recorded.

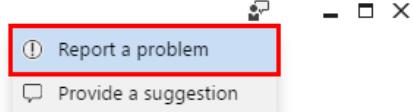
Submit the report

Choose the **Submit** button to send your report, along with any images and trace or dump files. (If the **Submit** button is grayed out, make sure that you've provided a title and description for the report.)

Reporting alternatives

Report a problem by using the Visual Studio Installer

If you can't complete the Visual Studio installation or you can't access the feedback tool within Visual Studio, you can report a problem by using the Visual Studio Installer. To do so, choose the feedback icon in the upper-right corner of the Visual Studio Installer.



Visual Studio Community 2017

Free, fully-featured IDE for students, open-source and individual developers

[License terms](#) | [Release notes](#)

(15.0.26228.4)

Welcome!

We invite you to go online to hone your skills and find additional tools to support your development workflow.

Learn

Whether you're new to development or an experienced developer, we have you covered with our tutorials, videos, and sample code.

Search for problems and solutions by using the Visual Studio developer community

If you don't want to or can't use Visual Studio to report a problem, there's a chance that the problem has already been reported and a solution posted in the Visual Studio developer community. For more information, see the [Visual Studio Developer Community](#) page.

Provide product feedback or a suggestion

If you don't have a problem to report but want to provide product feedback or a suggestion, there's a place for that, too. For more information, see the [UserVoice](#) page.

See Also

- [Talk to us](#)
- [Visual Studio developer community](#)

Resources for Troubleshooting Integrated Development Environment Errors

3/12/2018 • 1 min to read • [Edit Online](#)

Not all error messages have a specific associated Help topic. If the information in the error message does not help you resolve the problem, you can consult other resources such as Knowledge Base articles, the developer community, or product support.

Knowledge Base Articles

You can search the Knowledge Base (KB) online for articles about product issues. Not all issues have a corresponding KB article, but errors encountered by a significant number of customers are typically documented. You can access KB articles on the [Microsoft Support](#) Web site.

The Developer Community

Forums let you interact with other developers, and also Microsoft employees. If you encounter an error that you cannot find a resolution for, you can post questions about the issue on a forum. You can also search the newsgroups to see whether others have posted about the same issue.

You can access forums, blogs, chats, and other resources on the [Microsoft Technical Communities](#) Web site.

Product Support

If you still have questions after you try the other resources, you can contact Microsoft support services by visiting the [Microsoft Support](#) Web site. For information about product support available in your area, see [Talk to Us](#).

See also

- [Troubleshooting proxy errors](#)

Talk to Us

2/21/2018 • 2 min to read • [Edit Online](#)

We're interested in your ideas for improving our products and documentation. We can't always respond personally, but we'll make sure that your feedback gets to the right person or team.

I want to report a problem with Visual Studio

If you are running into issues using Visual Studio, such as crashes, sluggish performance, unexpected behavior, and so on, please report the problem to us by using the **Report a Problem** tool. Simply click the feedback icon next to **QuickLaunch**, or choose **Help > Send Feedback > Report a Problem** from the main menu. For more information, see [How to Report a Problem with Visual Studio 2017](#).

I want to make a suggestion about Visual Studio features

If there's something we can do better, please let us know! Choose **Help > Send Feedback > Provide a Suggestion** to suggest a feature or change to the Visual Studio team. Your suggestion will automatically be posted to [UserVoice](#), where others can up-vote it.

Rate this product

Pre-release versions of Visual Studio have a **Rate this Product** menu item that enables you to send feedback on the quality of the build you are running. You won't see this in released versions of the product.

I want to contact Microsoft Support

For Visual Studio support information, see the [Support Lifecycle & Servicing](#) page.

For other Microsoft products and services, see [Microsoft Support](#) for online help.

NOTE

Support outside the United States and Canada may vary. For a list of regional contacts, see [Microsoft Worldwide Sites](#).

For larger organizations that require managed support directly from Microsoft, contracts are available through various Enterprise Support offerings. For more information, see [Microsoft Enterprise Support Solutions](#).

If the product came installed with a new computer or device, the hardware manufacturer provides technical support and assistance for this software. Contact the manufacturer directly for support.

Microsoft support services are subject to then-current prices, terms, and conditions. Prices, terms, and conditions are subject to change without notice.

I want to get involved in the developer community

If you want to share your questions and answers with other developers, you can use the [Visual Studio Developer Community](#) site, the [MSDN Forums](#), or [StackOverflow](#). You can also view code from other developers and share your own examples on the [Developer code samples](#) site.

I want to help improve the Visual Studio documentation

Please use the **Was this page helpful?** feedback button at the bottom of the page. You can find this button on all

our documentation pages. Alternatively, you can add feedback to any Visual Studio page on docs.microsoft.com by using the **Sign in to give documentation feedback** button, also at the bottom of the page.

See Also

[How to Report a Problem with Visual Studio](#)