

CHAPTER 2



Bike Racer

SMALL VICTORIES CREATING C# PROJECTS

LEARNING OBJECTIVES

- *LIST AND DESCRIBE THE MINIMUM DEVELOPMENT TOOLS REQUIRED TO CREATE C# PROGRAMS*
- *STATE THE PURPOSE OF OPERATING SYSTEM ENVIRONMENT VARIABLES, INCLUDING THE PATH VARIABLE*
- *DEMONSTRATE YOUR ABILITY TO SET ENVIRONMENT VARIABLES IN WINDOWS 7 AND WINDOWS 10*
- *LIST AND DESCRIBE THE STEPS REQUIRED TO CREATE C# PROGRAMS FROM THE COMMAND PROMPT*
- *DEMONSTRATE YOUR ABILITY TO CREATE C# PROJECTS FROM THE COMMAND PROMPT*
- *DESCRIBE THE FUNCTIONS AND FEATURES GENERALLY FOUND IN AN INTEGRATED DEVELOPMENT ENVIRONMENT*
- *DEMONSTRATE YOUR ABILITY TO CREATE C# PROJECTS USING MICROSOFT VISUAL STUDIO COMMUNITY 2017*
- *DEMONSTRATE YOUR ABILITY TO USE THE NUGET PACKAGE MANAGER*
- *DEMONSTRATE YOUR ABILITY TO INSTALL MICROSOFT'S ROSLYN COMPILER TOOLS USING THE NUGET PACKAGE MANAGER*
- *EXPLAIN THE DIFFERENCE BETWEEN THE ROSLYN C# COMPILER AND THE .NET FRAMEWORK'S C# COMPILER*
- *LEARN HOW TO OBTAIN ADDITIONAL MICROSOFT DEVELOPMENT TOOLS FROM MICROSOFT IMAGINE*

INTRODUCTION

I call this chapter Small Victories because understanding your development tools and properly configuring your development environment easily accounts for seventy-five percent of the headaches you'll suffer when starting down the road of C# .NET development. Creating, compiling, and running your first program represents the biggest hurdle novice programmers face. The information in this chapter is designed to help you clear that hurdle and do a victory dance.

Here you will learn several critical software development skills. First, I explain exactly what you need to write, compile, and run C# programs. The good news is that you don't need a fortune to start programming with C# and the .NET Framework. Microsoft offers powerful software development tools absolutely free. Next, I will explain the purpose and use of operating system environment variables and show you how to configure the PATH environment variable so you can compile and run C# programs from the command prompt. And, since you may not be familiar with the command prompt, I will explain its purpose and demonstrate the use of several important commands.

Next, I'll show you how to create programs using Microsoft Visual Studio Community. Visual Studio Community is Microsoft's free Integrated Development Environment (IDE). An IDE increases programmer productivity by providing, under a common user interface, several important software development tools including source code editing, compiling, debugging, and execution profiling, and more.

Compiling C# Programs From The Command-Line

You only need two things to create professional, robust, C# programs: the Microsoft .NET Framework and a suitable text editor. Both can be obtained free of charge, although you will most likely want to buy a good text editor. I'm a big fan of Notepad++.

The .NET Framework supplies the C# compiler — `csc.exe`. The compiler transforms source code into Microsoft Intermediate Language (MSIL) modules which can be executed by the .NET Common Language Runtime (CLR).

You may be wondering, "Why?". "Why, if Microsoft offers Visual Studio, do I need to know how to create programs using the C# compiler from the command-line?" Well, that's a good question with several answers, and they go something like this: Visual Studio is a powerful program. In fact, it's so powerful that you can spend a lot of time just learning what it does and how to use it. So, in order to let you focus on learning the C# language, I recommend you learn how to use the command-line tools first and postpone your involvement with Visual Studio until after you've gained some programming experience.

My second answer to the question has a more practical side. You may have little or no experience using the command prompt. You may be familiar with the Windows interface, and with pointing and clicking a mouse, but you may have no experience issuing commands from the command-line.

I consider the ability to use the command prompt and to compile programs with the C# compiler from the command-line to be fundamental skills all programmers need to have in their tool belt. Mastering these skills will let you better understand what Visual Studio is doing under the covers. You may find it necessary one day to dive into the code generated automatically by Visual Studio to make a few adjustments. The only way you'll be able to do that is to take complete control of your development environment and understand how to use the command-line to compile and run C# programs.

DOWNLOAD AND INSTALL THE .NET FRAMEWORK

The first thing you need to do is to download and install the .NET Framework. Windows already comes with the .NET Framework installed, but it may not be the latest version. (**NOTE:** If you're installing .NET Framework 4.7.2 on Windows 7 make sure you've updated to Service Pack 1 (SP1)).

You will find the .NET Framework on Microsoft's .NET site: <https://www.microsoft.com/net>. Click the **Downloads** menu and select the **.NET Framework**. Download the latest edition of the **.NET Framework Developer Pack**, which, at the moment I write this is version 4.7.2.

At a minimum, you only need to download the .NET Framework. It provides the .NET runtime environment and the C# compiler, which supports the C# language up to version 5.0. If you want to target language features found in versions greater than 5.0, you'll need to download and install either the Roslyn compiler or Visual Studio Community 2017, which comes with the Roslyn compiler. I'll show you how to do both later in the chapter.

You can optionally download and install the Microsoft Windows Software Development Kit (SDK). The Microsoft Windows SDK provides additional development tools. However, before you can install the SDK, you must download and install the .NET Framework, or download the .NET Framework Developer Pack, which you can find with a quick Google search.

Installation of the .NET Framework is straightforward. The important thing to note during the installation process is where on your hard drive the .NET Framework is installed. The path to the .NET Framework installation directory will be `c:\Windows\Microsoft.NET\Framework64\`. Figure 2-1 shows the .NET Framework directory structure as it appears on my computer.

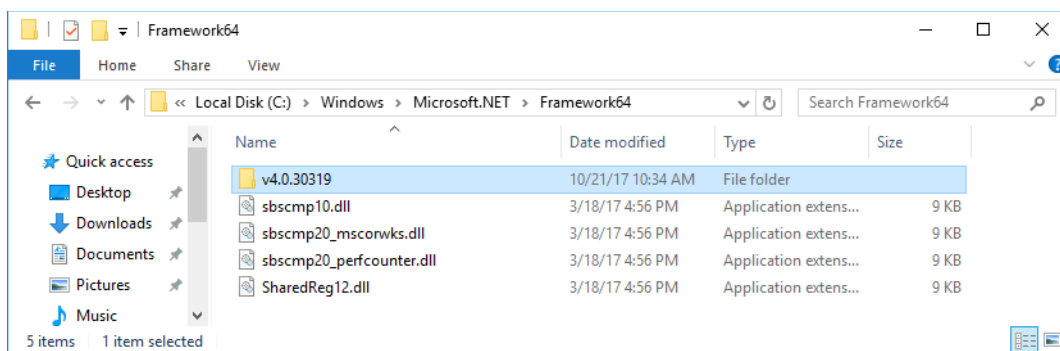


Figure 2-1: Microsoft.NET Framework Installation Directory

Referring to figure 2-1 — The C# compiler (`csc.exe`) resides in the `v4.0.30319` folder. Figure 2-2 shows a partial directory listing of the `v4.0.30319` folder. In there you'll find the C# compiler command-line tool.

Now that you've installed the .NET Framework, you have everything you need to compile and run C# programs. What you need now is a way to create C# source files, and for that you'll need a good text editor. I recommend Notepad++.

DOWNLOAD AND INSTALL NOTEPAD++

If you really wanted to rough-it you could use Notepad, the text editor that ships with Microsoft Windows. Notepad is perfectly suitable for creating small source files and lite editing jobs, but for programming projects, I recommend getting yourself a copy of Notepad++ from Notepad-Plus-Plus.org <http://notepad-plus-plus.org/>.

Installation of Notepad++ is quick and straightforward. During installation be sure to check the box to have a shortcut automatically installed on your desktop.

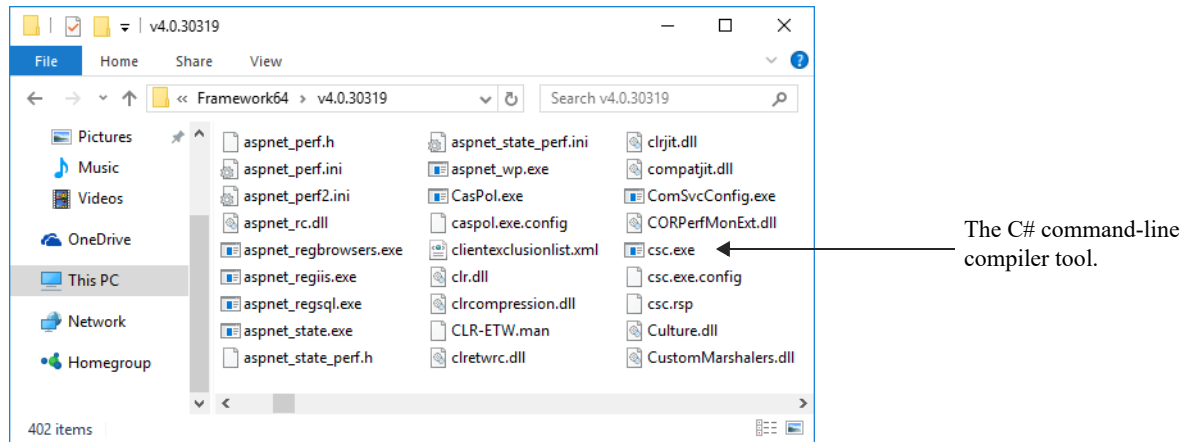


Figure 2-2: Partial Directory Listing of the v4.0.30319 Folder

Notepad++ is free to use, but if you like it and have the means, I recommend you make a small donation via the website to support its further development.

Armed now with the .NET Framework and a suitable text editor, you have everything you need to create, compile, and run C# programs. But before you get started you'll need to properly configure your development environment so that you can compile programs from the command-line.

CONFIGURING YOUR DEVELOPMENT ENVIRONMENT

A properly configured development environment is critical to the software creation process. In this section I will show you how to create and use operating system environment variables, how to create a project folder, how to set folder options so you can see filename suffixes, and how to set-up and configure shortcuts to the command console. I focus on two versions of Microsoft Windows, 7 and 10, and highlight the differences between the two where necessary. The skills you learn in this section will prove time and again to be absolutely invaluable throughout your software engineering career.

ENVIRONMENT VARIABLES

Environment variables are used to store data about the operating system environment. There are generally two types of environment variables: *system variables* and *user variables*.

System environment variables store data that pertains to and affects the operating system environment for all users. User environment variables store data that pertains to and affects the operating system environment for a particular user. Some system and user environment variables are automatically created and initialized by the operating system when it is installed, when applications are installed, and when users are created.

Several important environment variables must be created or edited before you can use the command-line tools to compile and run C# programs. These include: 1) a variable named DOT_NET_FRAMEWORK_HOME that contains the path to the installation location of the .NET Framework, and 2) the PATH variable that includes a reference to the DOT_NET_FRAMEWORK_HOME variable so the operating system knows where to find .NET-related executable files like the C# compiler (csc.exe).

CREATE ENVIRONMENT VARIABLES IN WINDOWS 7

The first environment variable you will set will be the location of the home directory of the .NET Framework. Navigate to that folder now so that you can copy the path to the .NET Framework directory; later, you will paste this value into the environment variable's value field. (Copying and pasting prevents

you from making mistakes when typing long path names) The path to this folder will be `c:\windows\Microsoft.NET\Framework64\v4.0.30319`. (Refer to figure 2-2) When you open the folder, select the path that shows up in the file explorer Address box and copy it using CTRL-C.

Next, you'll create the user environment variable named `DOT_NET_FRAMEWORK_HOME`. See figure 2-3 for an illustration of the complete environment variable creation process.

Right-click the **Computer** icon located on your desktop. If this icon is not located on your desktop, click the Windows Start button located in the lower left part of the taskbar, right-click **Computer** and click **Properties** from the pop-up to open the System control panel. In the System control panel click **Advanced system settings** located in the left-hand column to open the System Properties dialog.

In the System Properties dialog click the **Advanced** tab, then click the **Environment Variables** button to open the Environment Variables dialog window. Underneath the User variables section, click the **New** button to create a new environment variable. This will open the New User Variable dialog window. Enter `DOT_NET_FRAMEWORK_HOME` into the Variable name textbox. Paste the path to the .NET Framework home directory you copied earlier into the Variable value textbox. After entering both values, your New User Variable dialog window will look similar to the completed example shown in figure 2-3. Check your work for accuracy, then click the **OK** button to close the New User Variable dialog window. Click the **OK** button for each of the remaining open dialog windows to accept the changes. Congratulations! You just created an environment variable.

CREATE OR EDIT THE PATH ENVIRONMENT VARIABLE

Once you have the `DOT_NET_FRAMEWORK_HOME` environment variable set, you can use it to create or edit other environment variables. The next environment variable that you must either create or edit is the `PATH` variable. The operating system uses the `PATH` environment variable to locate executable files. An instance of the `PATH` (or Path) variable most likely already exists in the System Environment Variables section. I recommend leaving that version alone and creating another `PATH` environment variable in the User Environment Variables section. The two are combined to formulate the complete `PATH` value.

To create a new user `PATH` environment variable, follow the process illustrated in figure 2-3. Enter "PATH" into the Variable name text field. Enter the following into the Variable value text field: `%DOT_NET_FRAMEWORK_HOME%`. Click the **OK** buttons to accept the changes. (**Note:** To reference an environment variable's value, add a "%" to the beginning and end of the variable name.)

To edit an existing `PATH` environment variable, you'll need to select it and click the **Edit** button. Place your cursor in the Variable value text field and move to the far right end of the value that's entered there. If the existing value is not terminated with a semicolon, you'll need to add one before adding the `%DOT_NET_FRAMEWORK_HOME%` variable to the end like so:

preexisting path value;%DOT_NET_FRAMEWORK_HOME%

Figure 2-4 shows the `PATH` user environment variable being edited on my machine. Note the semicolons separating each environment variable in the `PATH`.

TEST NEWLY CREATED ENVIRONMENT VARIABLES

You can now check that you have set your environment variables correctly by running several tests. The first test entails opening a command prompt and using the `DOT_NET_FRAMEWORK_HOME` variable in a command. The second test is running the C# compiler from the command-line.

First, open a command prompt window. Do this by clicking on **Start->All Programs->Accessories->Command Prompt**. This will open a command prompt window like that shown in figure 2-5. Next, enter the following command at the command prompt: `cd %DOT_NET_FRAMEWORK_HOME%` The `cd` command stands for "Change Directory". If you have set the `DOT_NET_FRAMEWORK_HOME` environment vari-

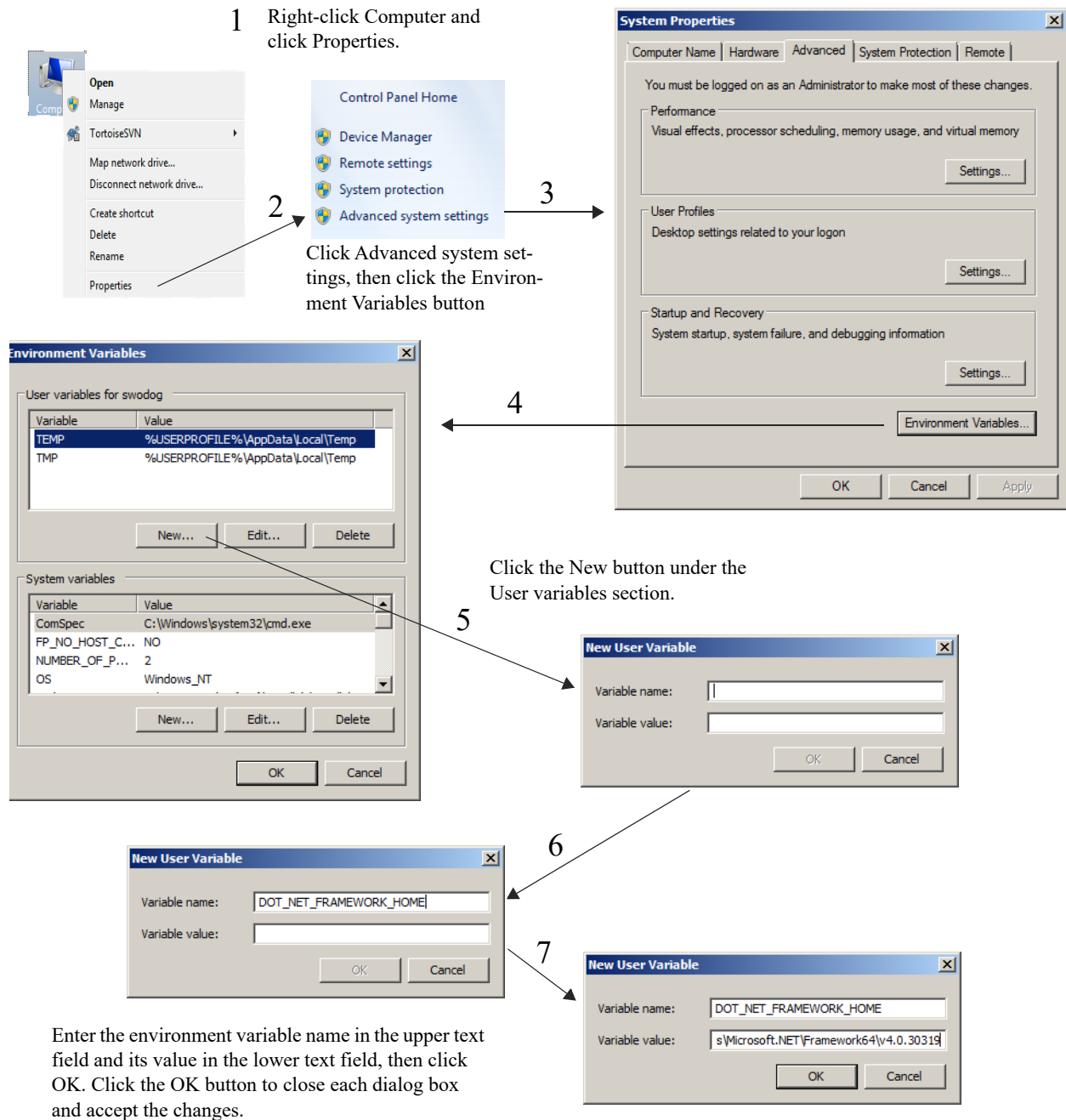


Figure 2-3: Creating an Environment Variable in Windows 7

able correctly, entering this command should take you to the `C:\Windows\Microsoft.NET\Framework64\v4.0.30319` directory as figure 2-6 illustrates.

Now test the PATH environment variable. Execute the following command in a command prompt window: `csc` This should run the C# compiler which will produce a result similar to that shown in figure 2-7. If your results look like those shown in figure 2-7, then you're good to go. Great job! If not, recheck your environment variable settings and try again until you have everything set just right.

Note that when you run the compiler (`csc.exe`) with no source file input you'll receive an error, like the one shown in figure 2-7. You'll also receive a message stating this version of the compiler only supports

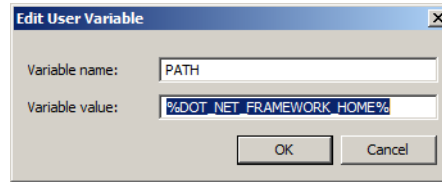


Figure 2-4: Editing the PATH User Environment Variable

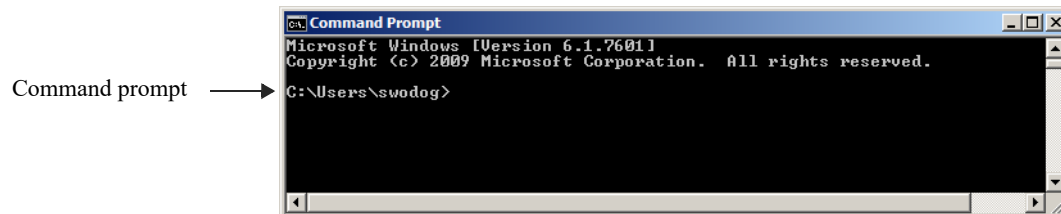


Figure 2-5: Command Prompt Window (A.K.A., Command Console)

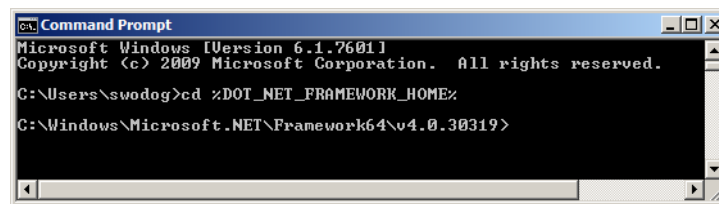


Figure 2-6: Testing the DOT_NET_FRAMEWORK_HOME Environment Variable

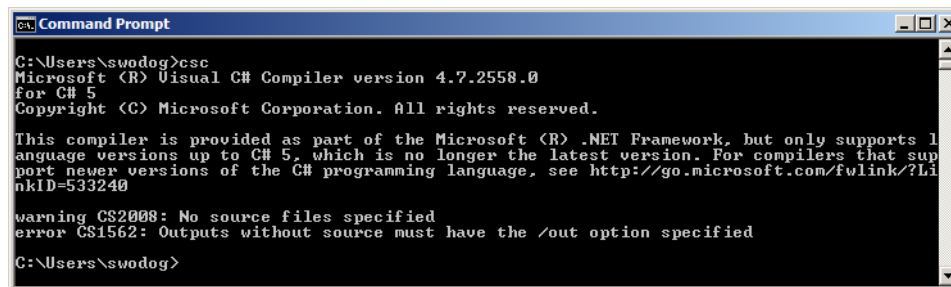


Figure 2-7: Testing the PATH Environment Variable by Running the C# Compiler

up to version 5 of the C# language. This will only be an issue if you try to use features found in language versions 6 or higher.

CREATING ENVIRONMENT VARIABLES IN WINDOWS 10

The steps required to create environment variables in Windows 10 are similar to Windows 7 with a few exceptions and icon name changes. Click the Windows Start button and click on the **File Explorer** Icon. Right-click on **This PC** in the left-hand panel and select **Properties** from the pop-up menu. This will take you to the System control panel as is shown in figure 2-8.

Referring to figure 2-8 — Click on **Advanced System Settings** located in the left hand panel. This will open the System Properties window. Click the **Environment Variables...** button to open the Environment Variables window. Under the User variables section click the **New...** button. This will open a New User Variable dialog box as is shown in figure 2.9.

Click Advanced system settings →

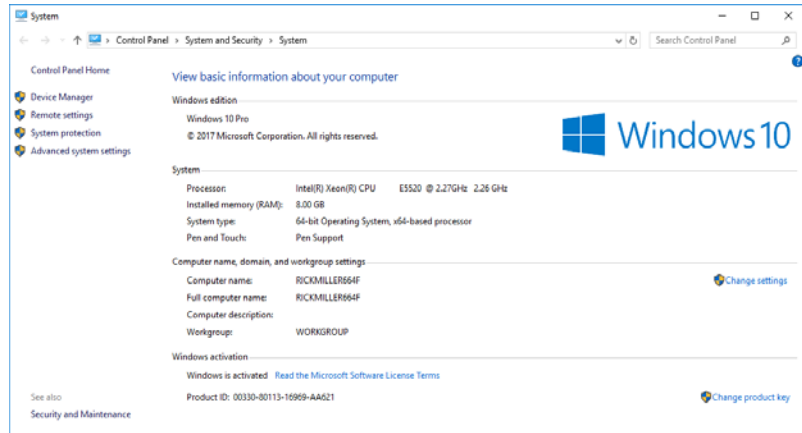


Figure 2-8: System Control Panel - Windows 10

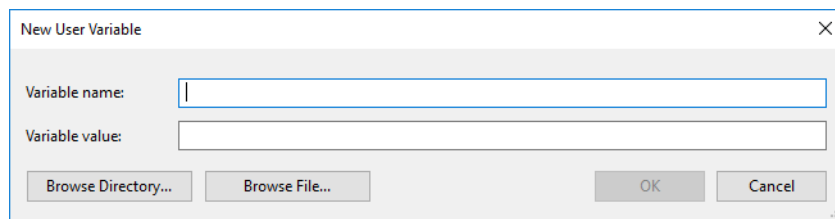


Figure 2-9: New User Variable Dialog - Windows 10

Referring to figure 2-9 — Enter the name of the environment variable in the Variable name box, and either paste the file path into the Variable value box or click the **Browse Directory...** button to locate the directory and have its path automatically entered into the box.

Next, select the user’s Path environment variable and click the **Edit...** button. This will open the Edit environment variable window as shown in figure 2-10.

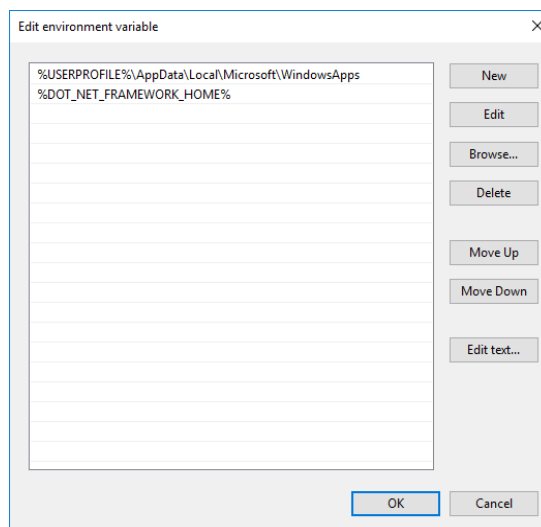


Figure 2-10: Editing Path Environment Variable in Windows 10

Referring to figure 2-10 — To add a new entry to the Path environment variable click the **New** button. This will allow entry into the next open field. Click **OK** to commit the edit and close the dialog window. Test your environment variables in Windows 10 using the command-line as explained in the previous section.

CREATE A PROJECTS FOLDER

The next thing you should do to set up your development environment is to create a folder named **Projects** in which to store your C# project folders and files. This folder will serve as the root folder for any individual projects you create. You will store each project in its own sub-folder under the Projects folder.

A good place to create the Projects folder is right on your desktop. To do this, right-click your desktop and select **New->Folder** from the pop-up menu as figure 2-11 illustrates.

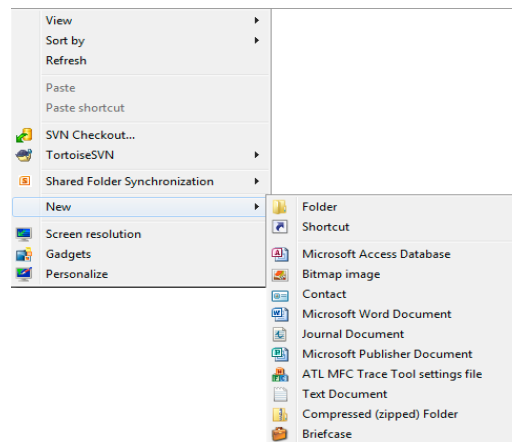


Figure 2-11: Creating a New Folder

Name the new folder “Projects”. Note that when you create a folder on your desktop, you are actually creating it in the [C:\Users\username\Desktop] folder, where “username” is the username of the account you used to log on to the computer. On my Windows 7 machine, the full path to the Projects folder created on the desktop is: [C:\Users\swodog\Desktop\Projects]

SET FOLDER OPTIONS IN WINDOWS 7

You’ll need to change folder options so you can see file-type extensions. Both novice and experienced programmers alike sometimes have difficulty trying to compile a C# file because the file they thought had an extension of “.cs” was in fact saved with an extension of “.cs.txt”, where the “.txt” extension was automatically added by a text editor, unbeknownst to the programmer. When this happens, the C# compiler will fail to recognize the file as a C# source file. To help prevent such headaches, it’s a great idea to change the folder options of all your folders to show file extensions.

To do this, open the Projects folder you just created and in it create a new text file. The easiest way to create the text file is to simply right-click in the open folder and select **New->Text Document** from the pup-up menu. Save the text document with the default name provided. Your Projects folder should now look like figure 2-12. Notice the name of the document you created simply shows as “New Text Document”. The “.txt” extension is hidden by default. So let’s unhide file extensions. Click the **Organize** drop-down menu and click **Folder and search options**. Click the **View** tab and scroll down until you see the check box that says, “Hide extensions for known file types”. This box is checked by default. Uncheck the box as is shown in figure 2-13.

Click the **Apply to All Folders** button in the Folder views section, then click the **Apply** button and lastly the **OK** button to dismiss the dialog. Your Projects folder should now look like figure 2-14.

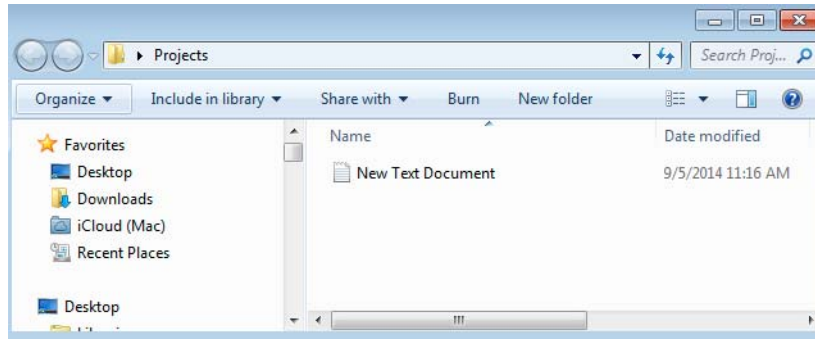


Figure 2-12: Projects Folder Before Setting Folder Options

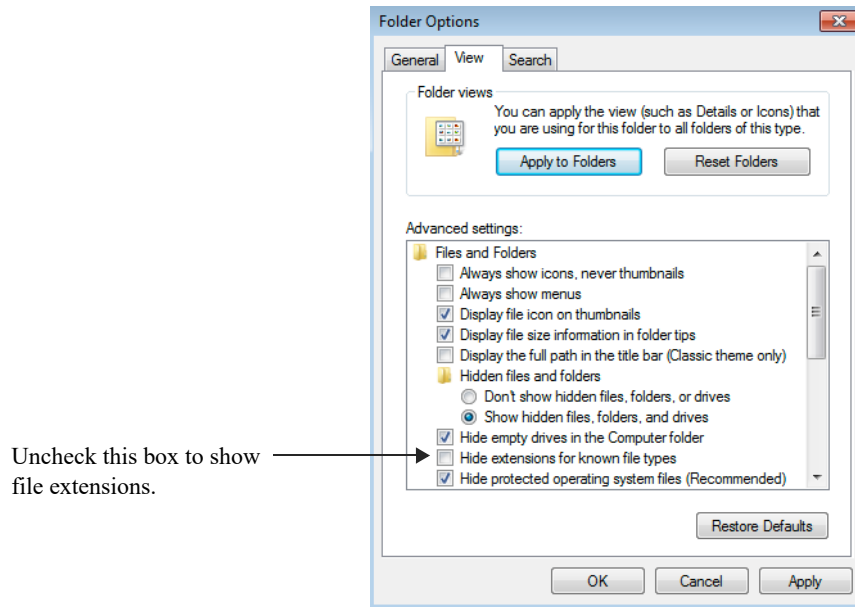


Figure 2-13: Folder Options Dialog Window

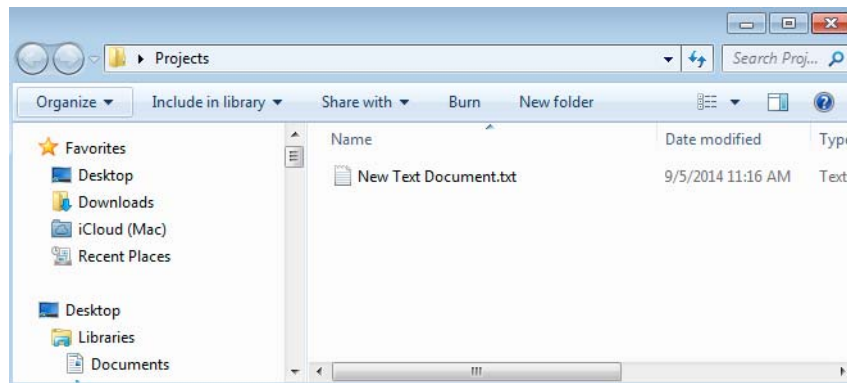


Figure 2-14: Projects Folder After Setting Folder Options

Notice now you can see the “.txt” file extension.

SET FOLDER OPTIONS IN WINDOWS 10

To set folder options in Windows 10, open a folder and click the **View** menu as is shown in figure 2-15.

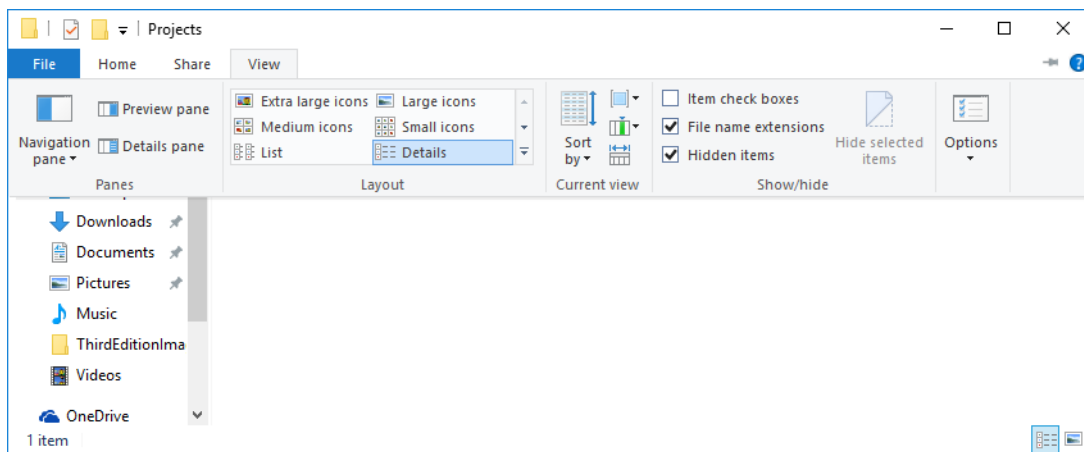


Figure 2-15: Setting Folder Options in Windows 10

Referring to figure 2-15 — In the Show/hide section check both the **File name extensions** and **Hidden items** check boxes.

CREATE A SHORTCUT TO THE COMMAND PROMPT AND SET ITS PROPERTIES

Since you'll be using the command prompt to compile C# programs you'll find it convenient to place a command prompt shortcut on your desktop. To do this in Windows 7, click **Start->All Programs->Accessories**. Right-Click Command Prompt and select **Send to... Desktop (Create Shortcut)**. This will create a new Command Prompt icon on the desktop named "Command Prompt".

In Windows 10 you can find the command prompt in the Windows System group under 'W' when you list **All apps**.

Test the shortcut by double-clicking it to open the command prompt window. By default, it should open to the directory `C:\Users\username`, where "username" is the account you used to log on to the computer. Figure 2-16 shows how the command prompt window looks with its default settings on my machine.

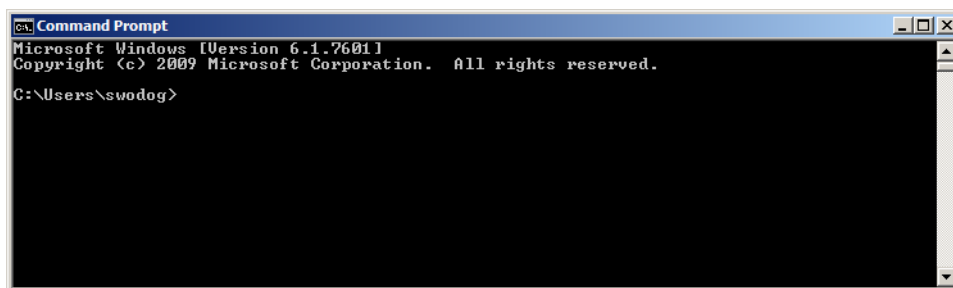


Figure 2-16: Default Command Console Window

CHANGE THE NAME OF THE COMMAND PROMPT SHORTCUT

The first bit of command prompt shortcut customization you'll want to do is to change its name. Do this by clicking twice on the shortcut icon's name, pausing between clicks longer than a standard double-

click. If you click too fast, you'll simply open the command prompt window. If this happens simply close the window and try again. Rename the shortcut to anything you want, but I recommend changing it to "Projects", or "C# Projects".

CHANGE THE STARTUP FOLDER SETTINGS WINDOWS 7 OR WINDOWS 10

Now that you've changed the shortcut's name, let's make a more meaningful change. It would be nice if the command prompt opened automatically in the Projects directory. To make this happen in Windows 7, right-click the command prompt shortcut icon located on your desktop and select **Properties**. This opens the properties dialog window for that shortcut. For example, if you renamed your shortcut to "Projects", the name of the properties dialog will be "Projects Properties". If you left the name of the shortcut with its default value, the name will be "Command Prompt Properties" as figure 2-17 illustrates. **Note:** The Shortcut tab is selected by default.

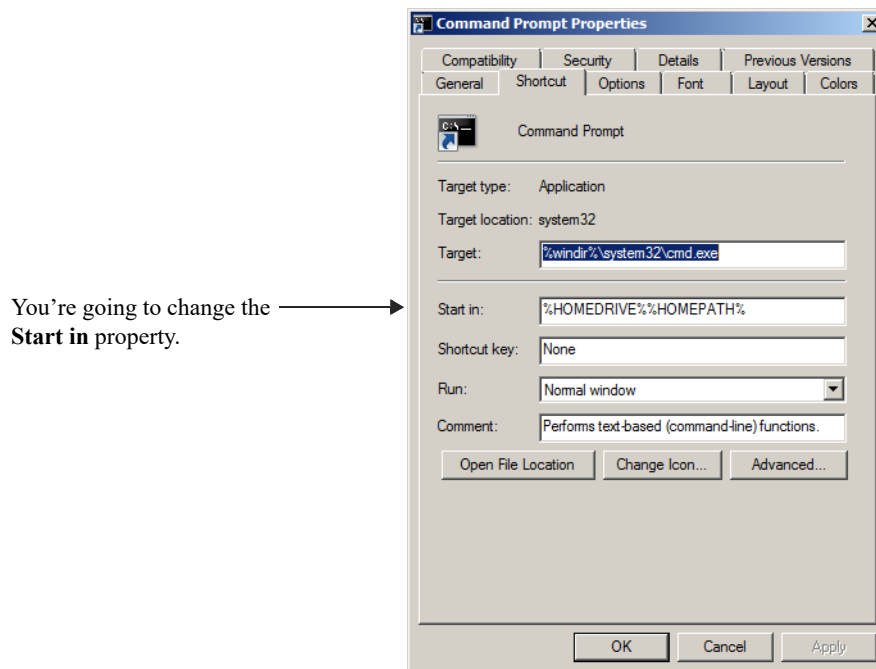


Figure 2-17: Command Prompt Properties Dialog

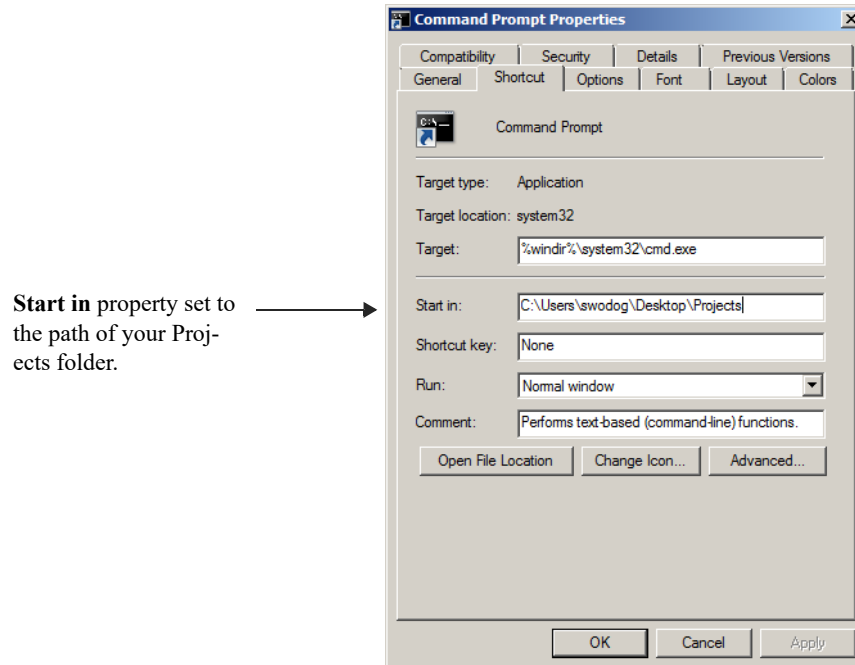
To make the command prompt automatically start in the Projects directory, change the **Start in** property by replacing its default contents with the full path to the Projects folder. If you placed your Projects folder on the desktop this will be `C:\Users\username\Desktop\Projects\`, where "username" is the account name you used to log on to the computer. Figure 2-18 shows the command prompt **Start in** property after I set it on my machine.

Click the **OK** button to accept the changes. Test the configuration by double-clicking the command prompt shortcut. It should open either in the Projects folder, or the folder you designated. If not, recheck your settings and try again until everything works as expected.

To do this in Windows 10 watch this video: <https://youtu.be/bVWY1KYxfVU>

CHANGE THE LAYOUT PROPERTIES

The last adjustment left to make to the command prompt shortcut is to change its default screen buffer size. This will allow you to increase the length and width of the command prompt window to see more information without the lines wrapping. Once again, right-click the command prompt shortcut and click

Figure 2-18: Setting the **Start in** Property

the **Properties** item to open the Properties dialog window. Click the **Layout** tab and set the screen buffer size **Width** property to 120, and change the **Height** property to 3000, as figure 2-19 illustrates.

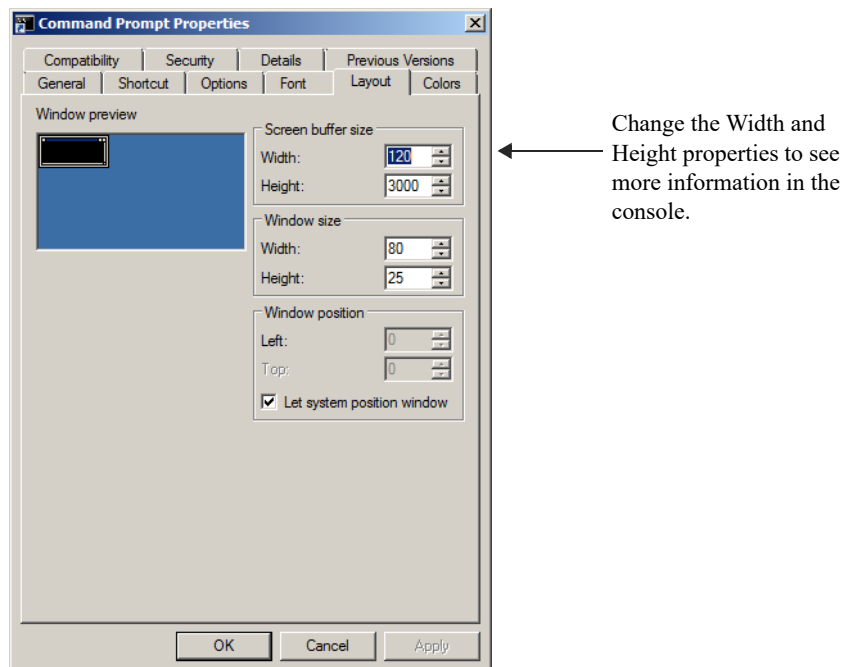


Figure 2-19: Setting Command Console Layout Properties

Click the **OK** button to accept the new changes. Now, double-click the command prompt shortcut to launch the command prompt. Change its height and width by dragging the lower right-hand corner. You'll find this to be a big help when troubleshooting and debugging your programs when you compile them from the command-line.

FINAL CONFIGURATION TEST

As a final test of the configuration, create and run a short C# program. To do this, you'll need to create the C# source file with the text editor, compile the source file using the C# command-line compiler, and then run the program.

CREATING THE SOURCE FILE

Using either Notepad++ or another text editor create a new file named "HelloWorld.cs" and save it in your Projects folder. Enter the code shown in example 2.1 into your source file and save the file.

2.1 HelloWorld.cs

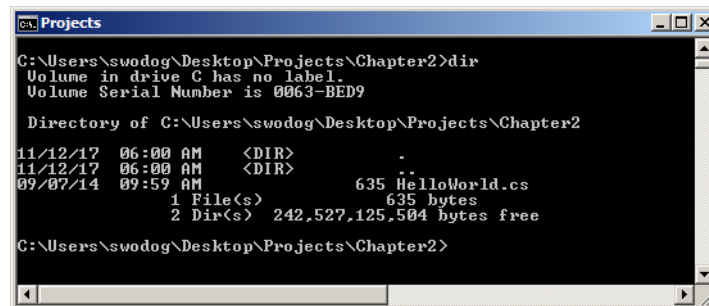
```

1      using System;
2
3      public class HelloWorld {
4
5          public static void Main(){
6
7              Console.WriteLine("Hello World!");
8          }
9      }

```

COMPILING THE SOURCE FILE

To compile the HelloWorld.cs file, open the command prompt and change to the directory where you saved the file. If you saved it in the Projects folder, then you're already there. If you created a sub-folder then change to that directory by using the `cd` command. For example, I saved the file in a folder named "Chapter2" located in the Projects folder. To change to the Chapter2 directory from the Projects directory I entered `cd chapter2`, then pressed the Return or Enter key. Figure 2-20 shows how the console looks on my machine when I use the `dir` command to list the directory contents.



```

C:\Users\swodog\Desktop\Projects\Chapter2>dir
Volume in drive C has no label.
Volume Serial Number is 0063-BED9

Directory of C:\Users\swodog\Desktop\Projects\Chapter2

11/12/17 06:00 AM <DIR>      .
11/12/17 06:00 AM <DIR>      ..
09/07/14 09:59 AM             635 HelloWorld.cs
               1 File(s)          635 bytes
               2 Dir(s)      242,527,125,504 bytes free

C:\Users\swodog\Desktop\Projects\Chapter2>

```

Figure 2-20: Directory Listing of the Chapter2 Directory Showing the HelloWorld.cs File

To compile the HelloWorld.cs file, enter `csc` followed by the name of the source file at the command prompt. If you entered the source code correctly, you should see results similar to those shown in figure 2-21.

If you execute another `dir` command to display the directory contents, you'll see a new file named "HelloWorld.exe". This is the executable program file.

EXECUTING THE APPLICATION

To run the executable file, simply enter its name at the command prompt. Figure 2-22 shows the results of running the Hello World program.

```

C:\Users\swodog\Desktop\Projects\Chapter2>csc HelloWorld.cs
Microsoft (R) Visual C# Compiler version 4.7.2558.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports la
hich is no longer the latest version. For compilers that support newer versions of the C# p
://go.microsoft.com/fwlink/?LinkID=533240

C:\Users\swodog\Desktop\Projects\Chapter2>

```

Figure 2-21: Compiling HelloWorld.cs Using the csc C# Compiler Command

Program output: →

```

C:\Users\swodog\Desktop\Projects\Chapter2>HelloWorld
Hello World!

C:\Users\swodog\Desktop\Projects\Chapter2>

```

Figure 2-22: Running the HelloWorld Program

FIXING COMPILER ERRORS

No matter how careful you try to be, you're bound to make a mistake or two (or more) when writing programs. Most of these mistakes will be simple typos, like forgetting to terminate a statement with a semi-colon, or misspelling the name of a variable, reserved keyword, etc. When you compile a program that contains a compiler error, you will see something similar to the output shown in figure 2-23.

```

C:\Users\swodog\Desktop\Projects\Chapter2>csc HelloWorld.cs
Microsoft (R) Visual C# Compiler version 4.7.2558.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only s
hich is no longer the latest version. For compilers that support newer versions o
://go.microsoft.com/fwlink/?LinkID=533240

HelloWorld.cs(15,39): error CS1002: ; expected

C:\Users\swodog\Desktop\Projects\Chapter2>

```

Figure 2-23: Compiler Output Showing Compiler Error on Line 6 at Position 39

When the compiler encounters a problem it will output one or more warning or error messages. Warnings are usually non-fatal, which means your program will still run if the compiler signals only a warning message. Error messages, on the other hand, must be addressed before your program will compile completely.

The error message will contain the name of the source file, the line number and character position of the problem, along with the compiler error code. The C# compiler error codes can be found on the Microsoft C# language reference site, but searching for them on Microsoft's website is tedious. The best way to find detailed information about a particular C# compiler error is to enter the following search query into Google: "C# compiler error CSNNNN", where "NNNN" is the compiler error number. The first result from this query will usually lead straight to the Microsoft C# compiler error page for that compiler error number. Figure 2-24 shows the detailed information page for Compiler Error CS1002.

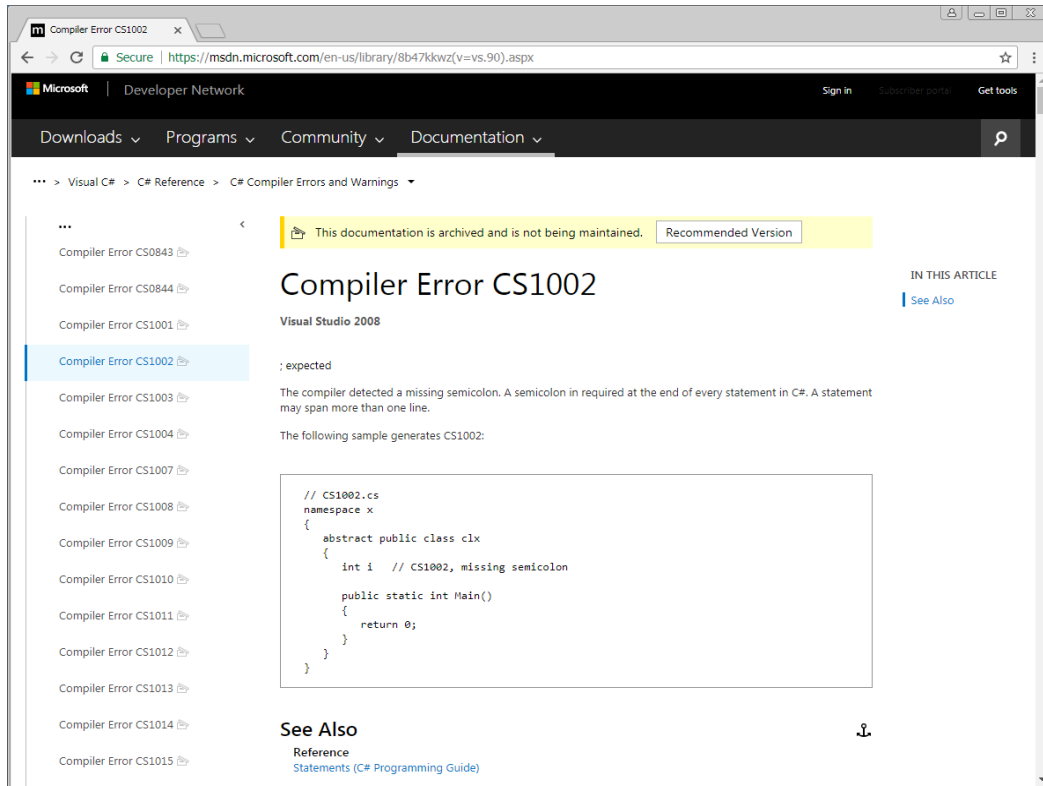


Figure 2-24: C# Compiler Error CS1002 “; expected”

Fix The First Compiler Error First

The best advice I can offer when dealing with compiler errors is to always **fix the first compiler error first**. The reason for this is that some compiler errors trigger other errors. Fixing the first error generally eliminates many other errors on the list.

Quick Review

All you need to create robust Microsoft C# applications is a good text editor and the C# compiler that’s included with the .NET Framework. Both can be obtained free of charge.

You must configure your development environment before you can compile programs from the command-line. This includes creating or editing one or more operating system environment variables. An environment variable is used to store data about the operating system environment. There are generally two types of environment variables: *system variables* and *user variables*. System environment variables store data that affects the operating system environment for all users; user environment variables store data that affects the operating system environment for a particular user.

Environment variable values can be accessed by enclosing the variable name in percent ‘%’ characters.

The operating system uses the PATH environment variable to locate executable files. You must create or edit the PATH environment variable to include the full path to the C# compiler (csc.exe).

It’s helpful to create a project folder and a shortcut to the command prompt on your desktop. Set the command prompt shortcut’s **Start in** property so it will automatically open in your designated projects folder. Increase the command prompt shortcut’s screen buffer **height** and **width** properties to see more information in the console window.

It's also a good idea to set your folder options to display file type extensions. This will prevent headaches associated with accidentally saving source files with a ".txt" extension.

To create a C# program you must first create the source file, compile the source file with the `csc` compiler tool, and then execute the program by typing its name at the command prompt and pressing the Return or Enter key.

You're bound to get a few compiler errors when you start writing your own programs. Use Google to search for the error code. This will lead you straight to the answer on Microsoft's documentation website. Remember to always **fix the first compiler error first!**

CREATING PROJECTS WITH MICROSOFT VISUAL STUDIO

Microsoft Visual Studio is an Integrated Development Environment (IDE) that combines text editing, project management, debugging, code profiling, and a host of other features, united by a common user interface. It lets you develop in a wide array of programming languages including C#, F#, Visual Basic, C++, Python, and R. With Visual Studio you can create classic Windows desktop applications and applications that run on Apple's OS X and Linux. You can do so much with the latest release of Visual Studio in general that the choices of languages and target technologies can be overwhelming.

Visual Studio comes in three flavors: Community, Professional, and Enterprise. Visual Studio Community has all the features necessary and important to individual developers, and while it doesn't have all the features of the Professional or Enterprise editions, what it does lack you may never notice. I recently led a team of talented software engineers building complex applications using Visual Studio Professional and we could have done everything just as easily with the Community edition because we didn't need or use the advanced features. So don't feel like you're settling for less if you don't have the Professional edition.

INSTALLING VISUAL STUDIO COMMUNITY 2017

Go to VisualStudio.com <http://visualstudio.com> and download the Visual Studio Community 2017 installer. The installation screen looks like that shown in figure 2-25.

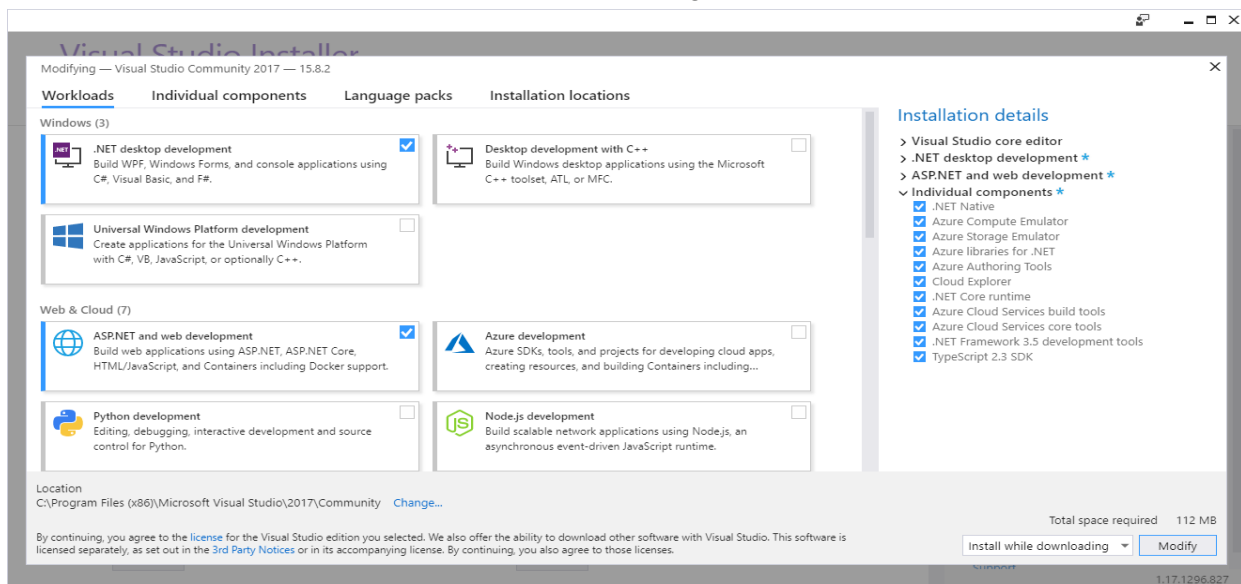


Figure 2-25: Visual Studio Community 2017 Installation Screen

Referring to figure 2-25 — When I wrote this, the current version was 15.8.2. Your version may be more recent. There are three tabs across the top: Workloads, Individual components, and Language packs. You can install whatever you like, but for this book you'll need the .NET desktop development, and ASP.NET and web development workloads. Under the Summary section, starting from the bottom, expand the ASP.NET and web development section and select all optional components with the exception of F#. Next, expand the .NET desktop development section and select all optional components except F# as is shown in figure 2-26.

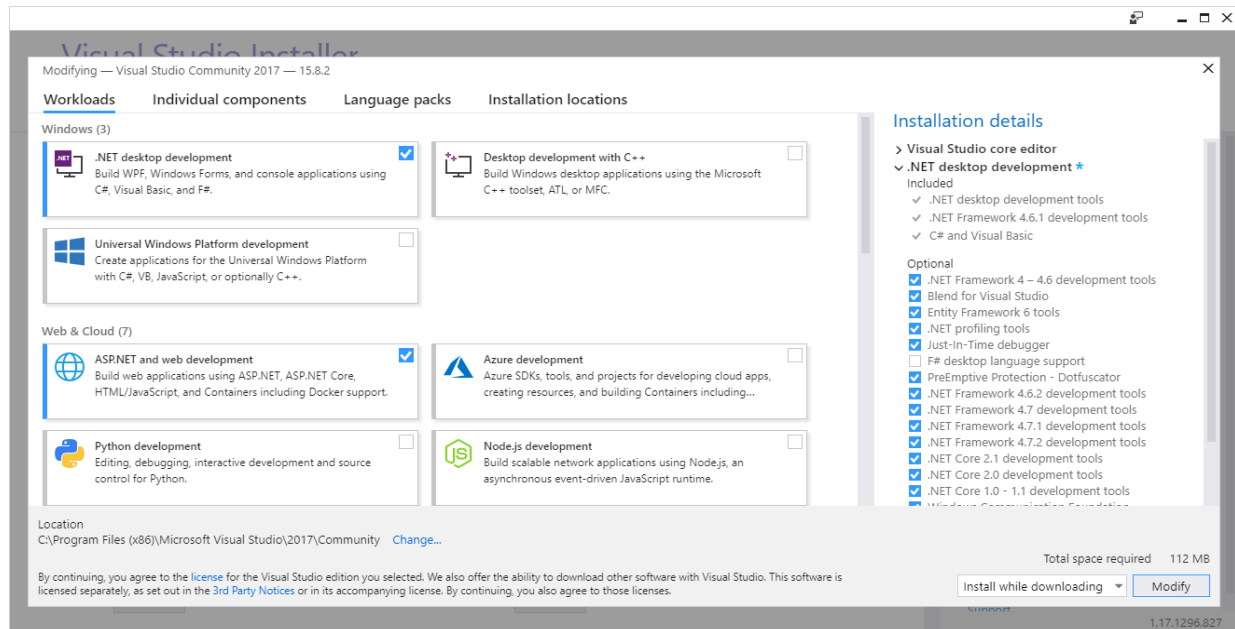


Figure 2-26: Selecting Optional Features and Components

Next, click the Individual components tab and under the .NET section check all the boxes as shown in figure 2-27.

That's all you need and most likely more than you need for the projects in this book. Click the Install (or Modify) button when you're ready. You can get up and get some coffee now while everything installs.

CREATING THE HelloWorld SOLUTION IN VISUAL STUDIO COMMUNITY 2017

When installation completes launch Visual Studio, complete the first startup tasks, and select **File -> New Project...** to open the New Project dialog as is shown in figure 2-28.

Referring to figure 2-28 — In the left panel expand the **Visual C#** section and select **Windows Desktop**. In the center panel select **Console App (.NET Framework)**. Down below, set the project **Name** to HelloWorld, set the **Location** to your Projects folder, and check the **Create directory for solution** box. Set the **Solution name** to HelloWorld as well. Check everything twice for good measure then click the **OK** button. Your HelloWorld project window will look similar to figure 2-29.

Referring to figure 2-29 — Note that the main window contains the code for a source file named Program.cs, which contains the definition for a class named Program. In the Solution Explorer panel located to the right you'll see a solution named HelloWorld which contains one project named HelloWorld.

The first order of business, although this is strictly not necessary, is to rename the Program.cs file to HelloWorld.cs. To do this, right-click the Program.cs file and select **Rename** from the pop-up menu. Rename the file HelloWorld.cs. When asked if you want to update references to the file click **Yes**. Your project window will now look like figure 2-30.

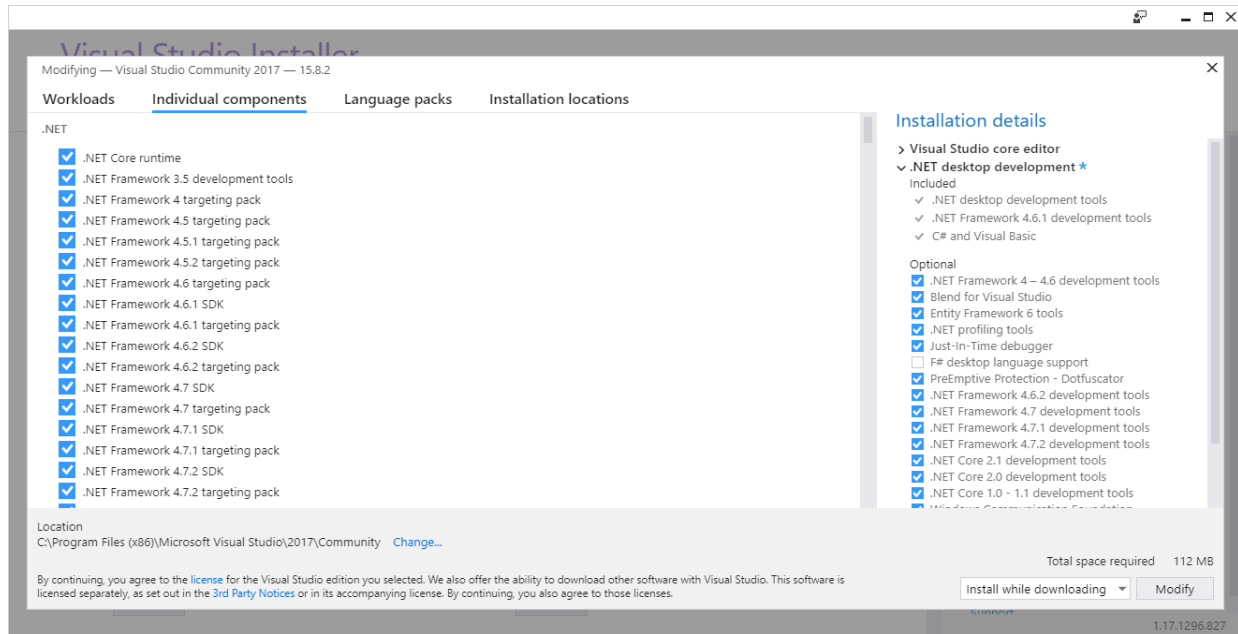


Figure 2-27: Selecting All Individual Components Under .NET Section

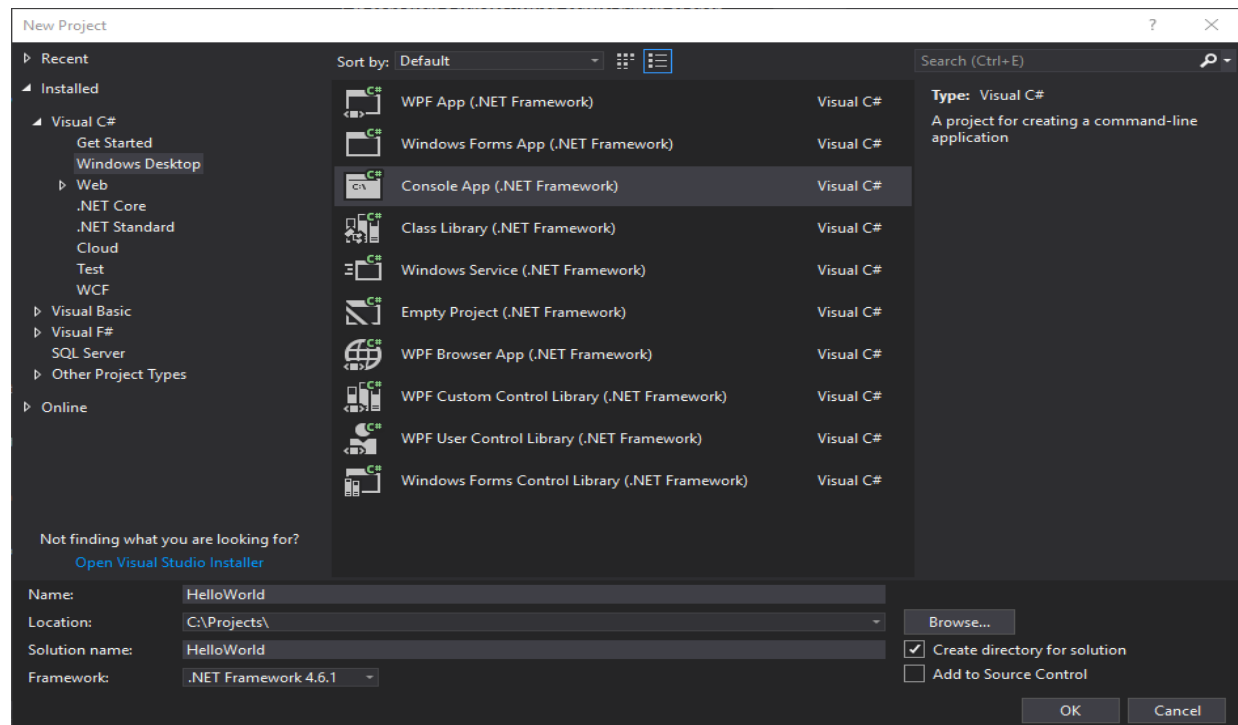


Figure 2-28: Creating New Visual C# Console App (.NET Framework)

Referring to figure 2-30 — Note the asterisk to the right of HelloWorld.cs tab. This means the file is not yet saved. Click the **disk icon** in the toolbar or use the key combination **CTRL-S** to save the file. You can see that Visual Studio has done a lot of the heavy lifting for you by way of creating boilerplate code. Note the using statements on lines 1 through 5. All we need is the `using System;` on line 1, but you can leave the others in as well. Note also that Visual Studio used a different version of the `Main()` method, spe-

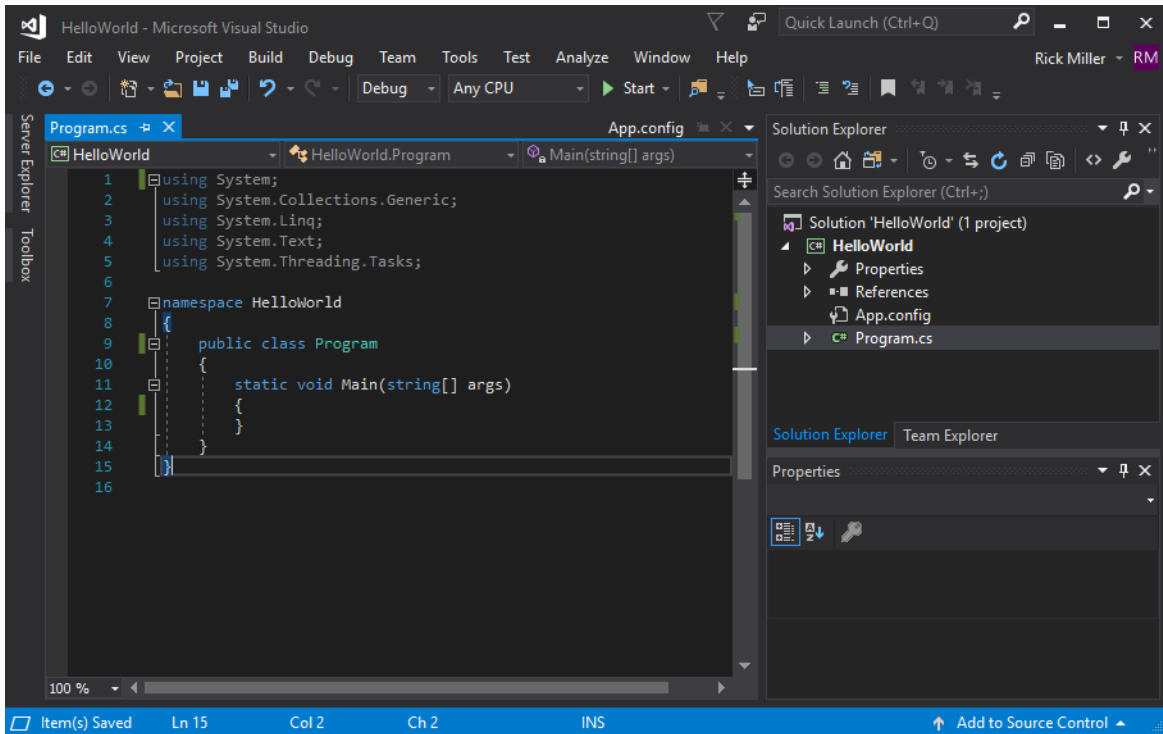


Figure 2-29: HelloWorld Project Window

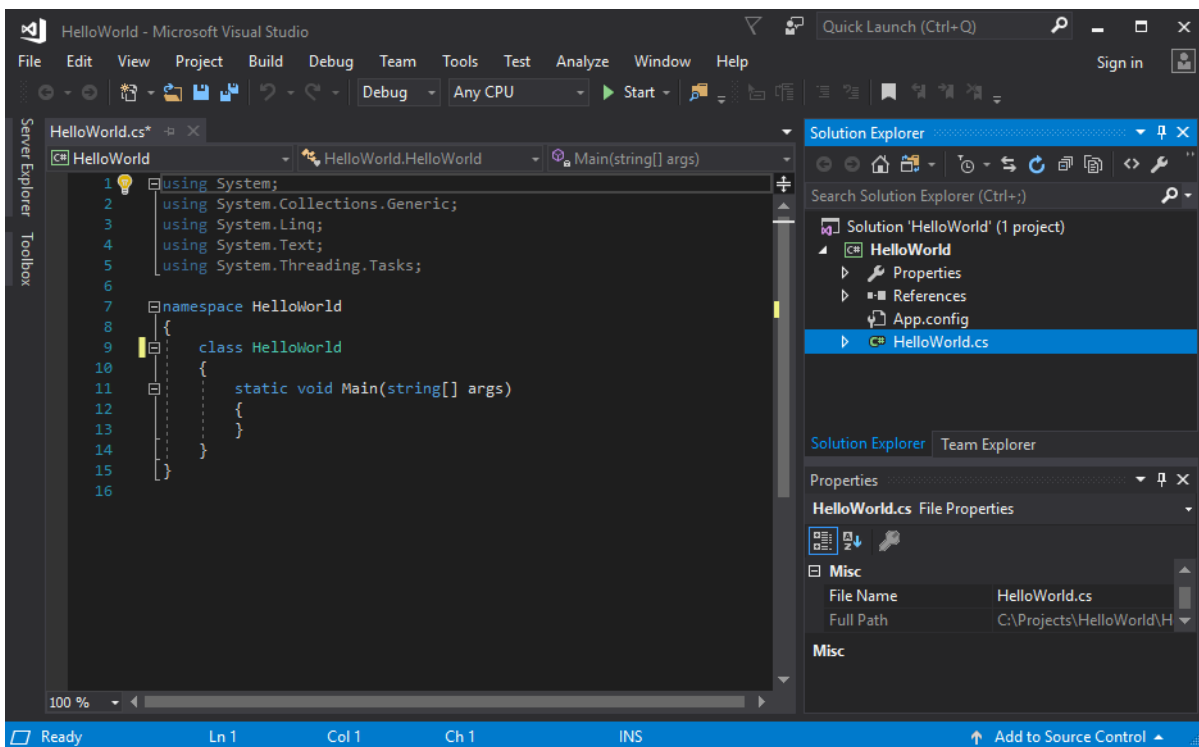


Figure 2-30: Renaming Program.cs to HelloWorld.cs

cifically, `Main(string[] args)`. You can leave this the way it is, too. All that's left to add to the `Main()` method is the following line of code: `Console.WriteLine("Hello World!");` as figure 2-31 shows.

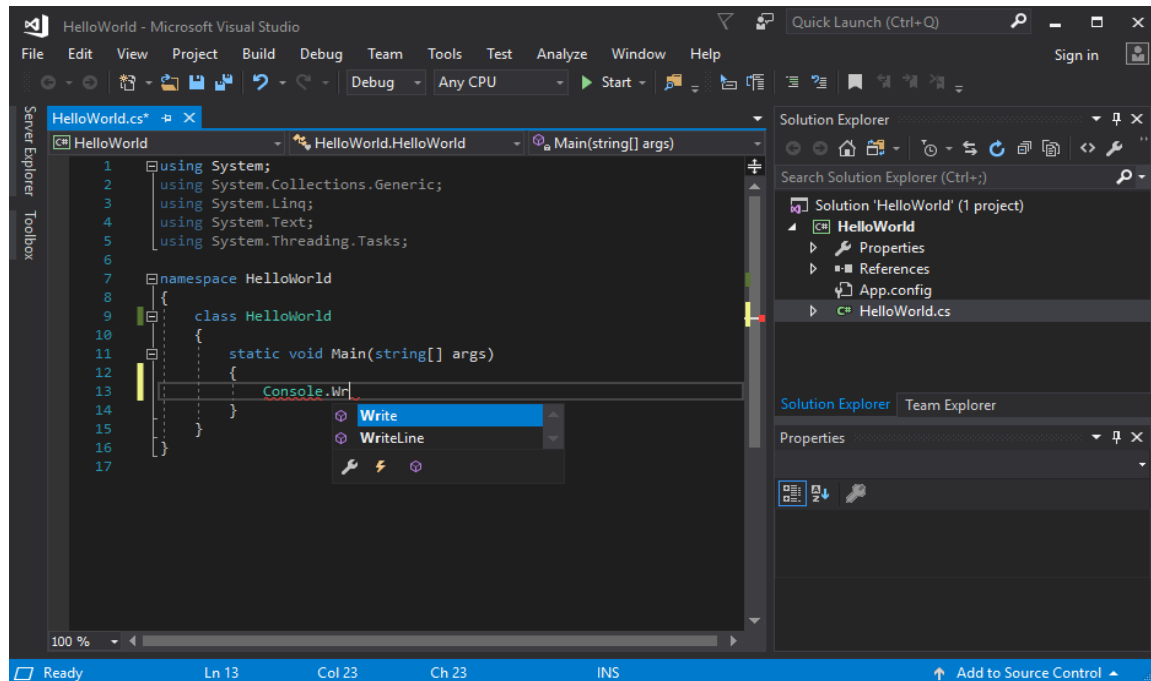


Figure 2-31: IntelliSense in Action

Referring to figure 2-31 — Notice that as you type, Visual Studio attempts to complete the line by showing what members (methods, properties, fields, etc.) are available. As you type `Console.Write`, IntelliSense offers you a choice between `Write` and `WriteLine`. You can hit the **down arrow** key to select `WriteLine` and then the **tab** key to complete the line with `WriteLine`. When you type the opening parenthesis '`(`', IntelliSense offers you a choice of overloaded methods to choose from as is shown in figure 2-32.

Referring to figure 2-32 — Notice there are 19 versions of the `WriteLine()` method. You'll learn more about method overloading later in the book. For now, complete the method by typing `"Hello World!"`. Don't forget the closing parenthesis `)` and the line-ending semicolon `;`. To run the program click **Start** in the toolbar next to the green triangle. If all goes well, the program will compile with no errors, and Visual Studio will launch the program. A black console window will appear briefly then disappear. The Output panel will open below the main program window as is shown in figure 2-33.

Referring to figure 2-33 — Note that you don't see the text `Hello World!` in the output window because it doesn't show console output. To run the program in such a way as to see the output you can do two things: 1) navigate to the project's output folder and run the program from there using the command prompt, or 2) add a line of code below the `Console.WriteLine()` method that pauses the program so you can see the output. I'll show you how to do both.

Running HelloWorld.exe From Project Output Directory

When you click **Start**, Visual Studio first builds the solution and the project it contains. In this case, it has only one project named `HelloWorld`. It then launches the program. In the case of a simple console application, it runs the program and immediately exits. That's why you only briefly see the black console window appear and then vanish. The results of building the `Hello World` project are contained in the project's `bin\Debug` folder as is shown in figure 2-34.

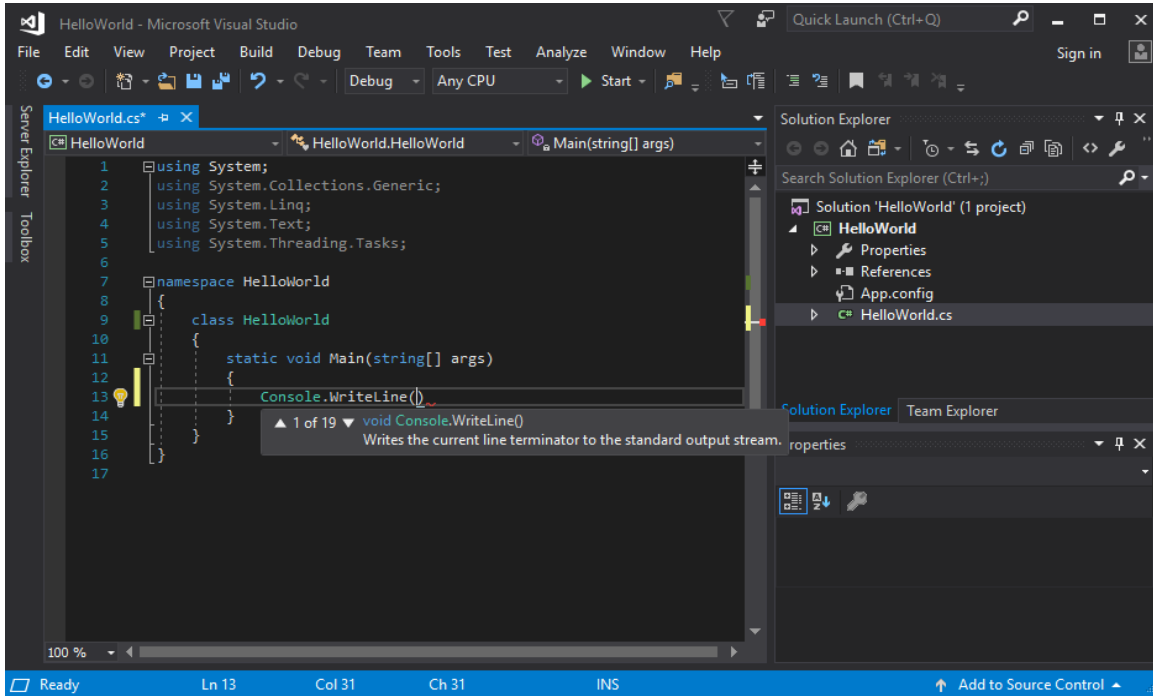


Figure 2-32: IntelliSense Showing Overloaded WriteLine() Methods

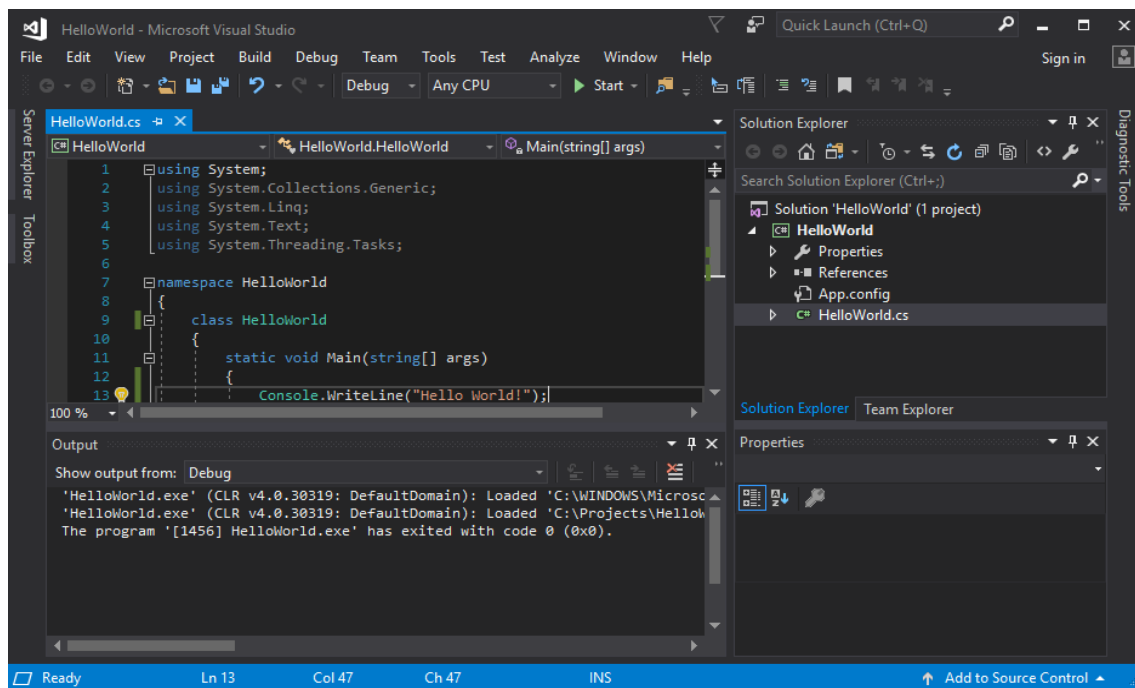


Figure 2-33: Output Panel Showing Results of Debug Session

Referring to figure 2-34 — Notice the HelloWorld.exe file. Open a command prompt and change directory to the HelloWorld project’s bin\debug folder. On my Windows 10 machine, the full path is: C:\Projects\HelloWorld\HelloWorld\bin\Debug

Type HelloWorld at the command prompt to run the program. Your results will look similar to figure 2-35.

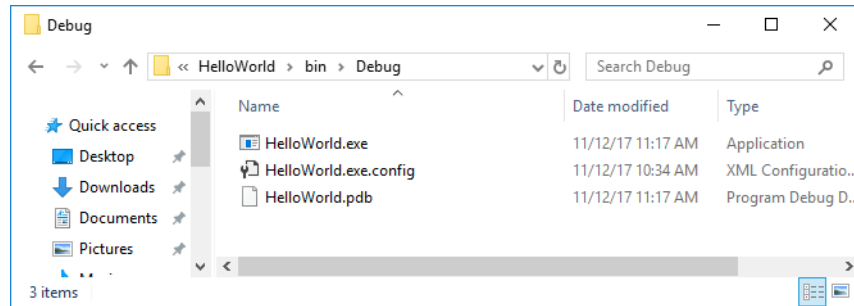


Figure 2-34: HelloWorld Project's bin\debug Folder

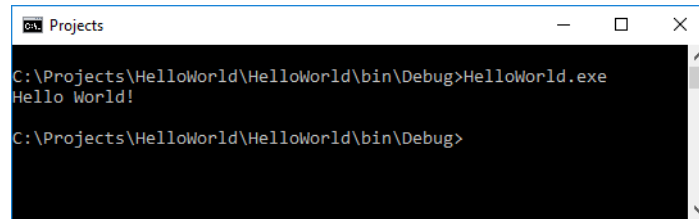


Figure 2-35: Running HelloWorld.exe via the Command Prompt from Debug Directory

PAUSING A CONSOLE PROGRAM TO SEE ITS OUTPUT

To pause the program briefly so you can see the results of running a console application launched from Visual Studio, add the following line of code to the HelloWorld program:

```
Console.ReadKey();
```

While you're at it, add the `public` keyword in front of the keyword `class` in the class definition on line 9. The source code for this version of HelloWorld.cs will look like example 2.2.

2.2 HelloWorld.cs (Modified)

```

1     using System;
2     using System.Collections.Generic;
3     using System.Linq;
4     using System.Text;
5     using System.Threading.Tasks;
6
7     namespace HelloWorld
8     {
9         public class HelloWorld
10        {
11            static void Main(string[] args)
12            {
13                Console.WriteLine("Hello World!");
14                Console.ReadKey();
15            }
16        }
17    }

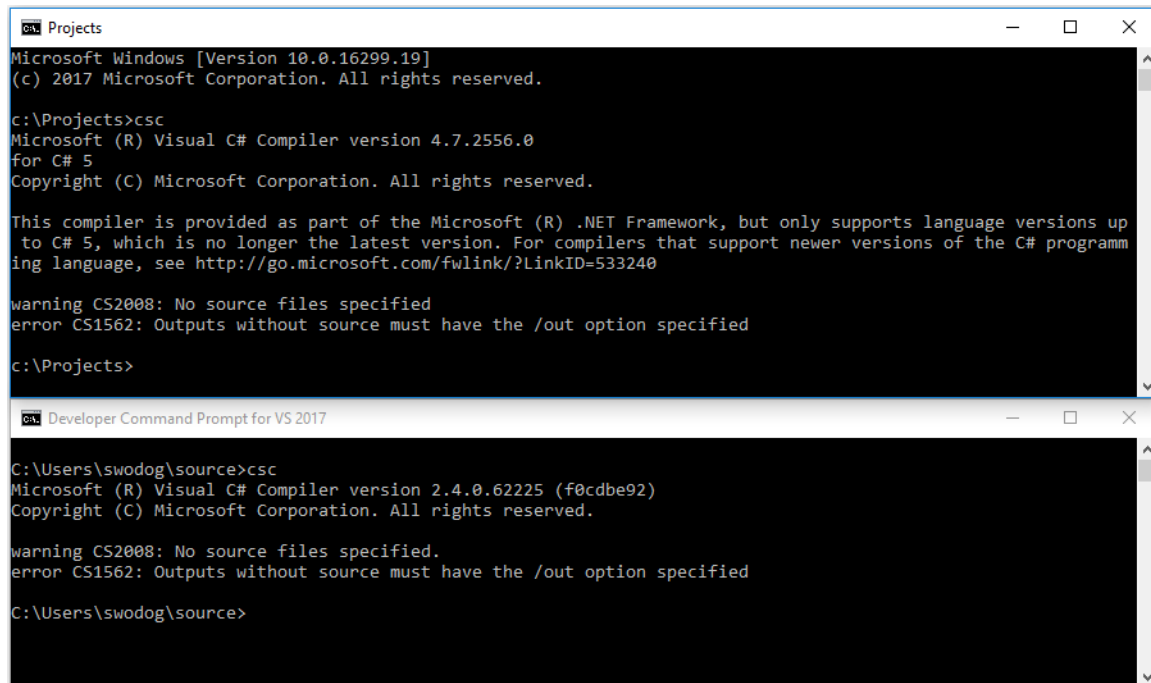
```

Referring to example 2.2 — When you run this version, the console will pause until you press a key on the keyboard.

Visual Studio is a powerful IDE and this short demo didn't even begin to highlight its significant capabilities. I still strongly recommend learning how to create and compile C# programs using only Notepad++ and the C# compiler tool, even if that route makes you think harder and creates a little more struggle on your part. In this case, a little pain creates a lot of gain.

THE ROSLYN COMPILER

If you installed Visual Studio Community 2017 you have access to the Roslyn compiler, which supports the C# language up to and including versions 6 and above. No need to set any environment variables, just find the Developer Command Prompt for VS 2017. It will be in the Visual Studio 2017 folder in Windows 7 and 10. Launch the Developer Command Prompt and the Projects command prompt you created earlier and compare them side-by-side as is shown in figure 2-36.



```
Microsoft Windows [Version 10.0.16299.19]
(c) 2017 Microsoft Corporation. All rights reserved.

c:\Projects>csc
Microsoft (R) Visual C# Compiler version 4.7.2556.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework, but only supports language versions up
to C# 5, which is no longer the latest version. For compilers that support newer versions of the C# program
ing language, see http://go.microsoft.com/fwlink/?LinkID=533240

warning CS2008: No source files specified
error CS1562: Outputs without source must have the /out option specified

c:\Projects>
```

```
C:\Users\swodog\source>csc
Microsoft (R) Visual C# Compiler version 2.4.0.62225 (f0cdb92)
Copyright (C) Microsoft Corporation. All rights reserved.

warning CS2008: No source files specified.
error CS1562: Outputs without source must have the /out option specified

C:\Users\swodog\source>
```

Figure 2-36: Projects Command Prompt Above; Developer Command Prompt Below

Referring to figure 2-36 — Note that the compiler tool provided with the .NET Framework version 4.7.1 is version 4.7.2556.0, while the Roslyn compiler is version 2.4.0.62225. Even I find this numbering to be somewhat confusing. Bottom line, if I introduce a C# language feature that requires the Roslyn compiler, I'll highlight that fact in the text and give you step-by-step instructions on how to compile the code.

WHAT IS THE ROSLYN COMPILER?

Roslyn is Microsoft's codename for the .NET Compiler Platform, first introduced around 2011. Roslyn is actually a set of open-source C# and Visual Basic compilers that make it easier for developers to write advanced code analysis tools. Essentially, the Roslyn compilers integrate better with Visual Studio and support advanced features offered by C# language versions 6 and above.

A detailed discussion of the differences between the C# compiler that ships with the .NET Framework and the Roslyn compiler is beyond the scope of this book. For more information about the .NET Compiler Platform visit the Roslyn GitHub site: <https://github.com/dotnet/roslyn>

The important points to note about Roslyn are that it's open source and you can download it free from the GitHub site. I show you how to install and use the Roslyn compiler from the command-line using the NuGet package manager in the next section.

Quick Review

Visual Studio is a powerful Integrated Development Environment (IDE) that combines editing, compiling, debugging, code profiling, and other services. Visual Studio Community is freely available from Microsoft.

Even though Visual Studio is a powerful tool, it will serve you well to learn how to compile C# programs from the command-line. This will enable you to better understand what's happening under the covers and let you fix things when something goes wrong.

If you install Visual Studio the Roslyn compilers are already installed. To use the Roslyn compiler from the command-line simply open the Visual Studio Developer Command Prompt.

INSTALLING THE ROSLYN COMPILER STAND-ALONE

This section explains how to download, install, and use the Roslyn compiler from the command-line. The overall process goes like this:

- Download and install the NuGet package manager
- Use the NuGet package manager to download and install the Roslyn compilers
- Configure environment variables to point to the Roslyn compiler installation location
- Test the configuration by running `csc.exe` from the command-line

DOWNLOAD AND INSTALL THE NUGET PACKAGE MANAGER

Go to <https://www.nuget.org/downloads> and download the latest release of the NuGet package manager `nuget.exe`. I recommend putting the executable in the following directory, which you'll need to create:

```
c:\DevTools\NuGet
```

Open a command prompt and change to the NuGet installation directory and type `nuget` to get a list of help commands. Your console output should look similar to figure 2-37.

Referring to figure 2-37 — You'll use the NuGet `install` command to install the `Microsoft.Net.Compilers` package. At the command prompt type `nuget install /help` to get detailed instructions on how to use the NuGet `install` command as is shown in figure 2-38.

Referring to figure 2-38 — Use the `-OutputDirectory` or `-o` option to specify the installation directory.

USE THE NUGET PACKAGE MANAGER TO DOWNLOAD AND INSTALL ROSLYN

I recommend installing the compilers in the following directory:

```
c:\DevTools\Roslyn
```

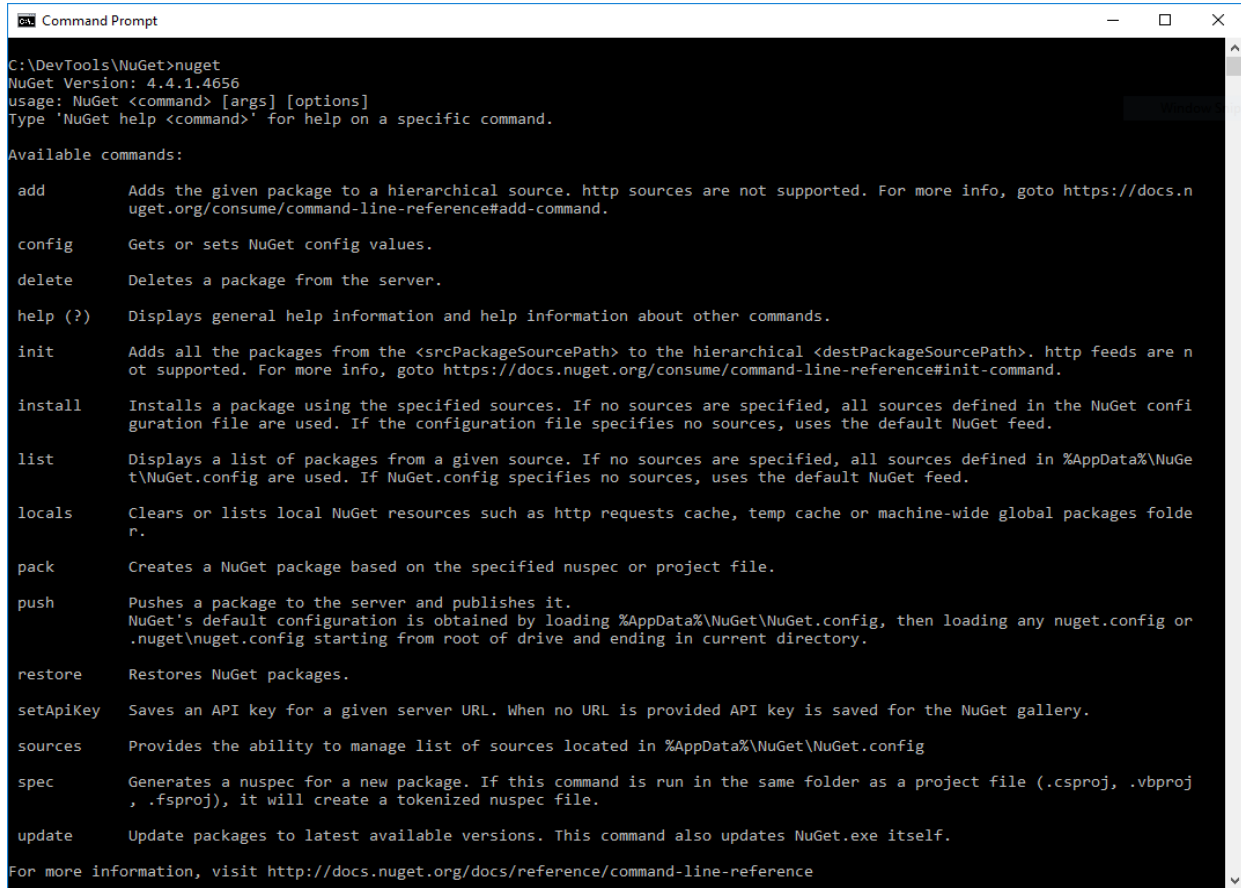
Type the following NuGet command at the command prompt to install the most current version of Roslyn compilers into this directory:

```
nuget install Microsoft.Net.Compilers -o c:\DevTools\Roslyn
```

Your results should look similar to figure 2-39 depending on the version of the `Microsoft.Net.Compilers` installed.

Referring to figure 2-39 — I recommend inspecting the installation directory to see what was installed and where. Figure 2-40 shows the contents of the installation directory on my machine.

Referring to figure 2-40 — Note that the `csc.exe` file is located in the `C:\DevTools\Roslyn\Microsoft.Net.Compilers.2.4.0\tools` directory.



```

C:\DevTools\NuGet>nuget
NuGet Version: 4.4.1.4656
Usage: NuGet <command> [args] [options]
Type 'NuGet help <command>' for help on a specific command.

Available commands:

add          Adds the given package to a hierarchical source. http sources are not supported. For more info, goto https://docs.nuget.org/consume/command-line-reference#add-command.

config       Gets or sets NuGet config values.

delete       Deletes a package from the server.

help (?)     Displays general help information and help information about other commands.

init         Adds all the packages from the <srcPackageSourcePath> to the hierarchical <destPackageSourcePath>. http feeds are not supported. For more info, goto https://docs.nuget.org/consume/command-line-reference#init-command.

install      Installs a package using the specified sources. If no sources are specified, all sources defined in the NuGet configuration file are used. If the configuration file specifies no sources, uses the default NuGet feed.

list         Displays a list of packages from a given source. If no sources are specified, all sources defined in %AppData%\NuGet\NuGet.config are used. If NuGet.config specifies no sources, uses the default NuGet feed.

locals      Clears or lists local NuGet resources such as http requests cache, temp cache or machine-wide global packages folder.

pack         Creates a NuGet package based on the specified nuspec or project file.

push         Pushes a package to the server and publishes it. NuGet's default configuration is obtained by loading %AppData%\NuGet\NuGet.config, then loading any nuget.config or .nuget\NuGet.config starting from root of drive and ending in current directory.

restore      Restores NuGet packages.

setApiKey    Saves an API key for a given server URL. When no URL is provided API key is saved for the NuGet gallery.

sources     Provides the ability to manage list of sources located in %AppData%\NuGet\NuGet.config

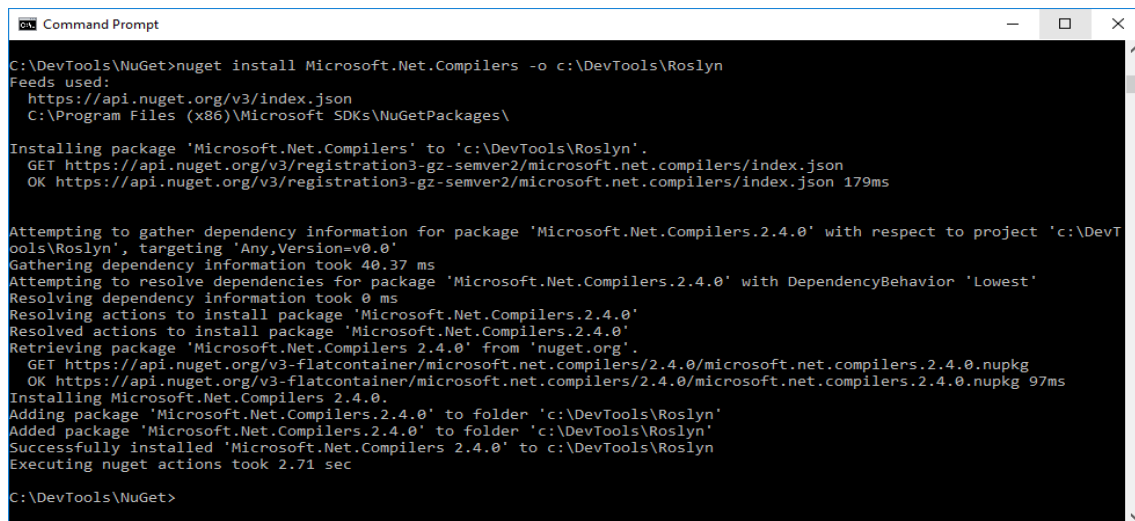
spec        Generates a nuspec for a new package. If this command is run in the same folder as a project file (.csproj, .vbproj, .fsproj), it will create a tokenized nuspec file.

update      Update packages to latest available versions. This command also updates NuGet.exe itself.

For more information, visit http://docs.nuget.org/docs/reference/command-line-reference

```

Figure 2-37: Type nuget with no Arguments to get List of Commands



```

C:\DevTools\NuGet>nuget install Microsoft.Net.Compilers -o c:\DevTools\Roslyn
Feeds used:
  https://api.nuget.org/v3/index.json
  C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\

Installing package 'Microsoft.Net.Compilers' to 'c:\DevTools\Roslyn'.
  GET https://api.nuget.org/v3/registration3-gz-semver2/microsoft.net.compilers/index.json
  OK https://api.nuget.org/v3/registration3-gz-semver2/microsoft.net.compilers/index.json 179ms

Attempting to gather dependency information for package 'Microsoft.Net.Compilers.2.4.0' with respect to project 'c:\DevTools\Roslyn', targeting 'Any,Version=v0.0'
Gathering dependency information took 40.37 ms
Attempting to resolve dependencies for package 'Microsoft.Net.Compilers.2.4.0' with DependencyBehavior 'Lowest'
Resolving dependency information took 0 ms
Resolving actions to install package 'Microsoft.Net.Compilers.2.4.0'
Resolved actions to install package 'Microsoft.Net.Compilers.2.4.0'
Retrieving package 'Microsoft.Net.Compilers 2.4.0' from 'nuget.org'.
  GET https://api.nuget.org/v3-flatcontainer/microsoft.net.compilers/2.4.0/microsoft.net.compilers.2.4.0.nupkg
  OK https://api.nuget.org/v3-flatcontainer/microsoft.net.compilers/2.4.0/microsoft.net.compilers.2.4.0.nupkg 97ms
Installing Microsoft.Net.Compilers 2.4.0.
Adding package 'Microsoft.Net.Compilers.2.4.0' to folder 'c:\DevTools\Roslyn'
Added package 'Microsoft.Net.Compilers.2.4.0' to folder 'c:\DevTools\Roslyn'
Successfully installed 'Microsoft.Net.Compilers 2.4.0' to c:\DevTools\Roslyn
Executing nuget actions took 2.71 sec

C:\DevTools\NuGet>

```

Figure 2-39: Results of Installing Microsoft.Net.Compilers NuGet Package

```

C:\DevTools\NuGet>nuget install /help
usage: NuGet install packageId|pathToPackagesConfig [options]

Installs a package using the specified sources. If no sources are specified, all sources defined in the NuGet configuration file are used. If the configuration file specifies no sources, uses the default NuGet feed.

Specify the id and optionally the version of the package to install. If a path to a packages.config file is used instead of a n id, all the packages it contains are installed.

options:
-OutputDirectory          Specifies the directory in which packages will be installed. If none specified, uses the current directory.
-Version                 The version of the package to install.
-DependencyVersion       Overrides the default dependency resolution behavior.
-Framework               Target framework used for selecting dependencies. Defaults to 'Any' if not specified.
-ExcludeVersion          (x) If set, the destination folder will contain only the package name, not the version number
-Prerelease              Allows prerelease packages to be installed. This flag is not required when restoring packages by installing from packages.config.
-RequireConsent          Checks if package restore consent is granted before installing a package.
-SolutionDirectory       Solution root for package restore.
-Source +                A list of packages sources to use for this command.
-FallbackSource +        A list of package sources to use as fallbacks for this command.
-NoCache                 Disable using the machine cache as the first package source.
-DirectDownload          Download directly without populating any caches with metadata or binaries.
-DisableParallelProcessing Disable parallel processing of packages for this command.
-PackageSaveMode         Specifies types of files to save after package installation: nuspec, nupkg, nuspec;nupkg.
-Help                   (?) help
-Verbosity               Display this amount of details in the output: normal, quiet, detailed.
-NonInteractive          Do not prompt for user input or confirmations.
-ConfigFile              The NuGet configuration file. If not specified, file %AppData%\NuGet\NuGet.config is used as configuration file.
-ForceEnglishOutput      Forces the application to run using an invariant, English-based culture.

examples:
nuget install elmah
nuget install packages.config
nuget install ninject -o c:\foo

For more information, visit http://docs.nuget.org/docs/reference/command-line-reference

C:\DevTools\NuGet>

```

Figure 2-38: NuGet install Command Help

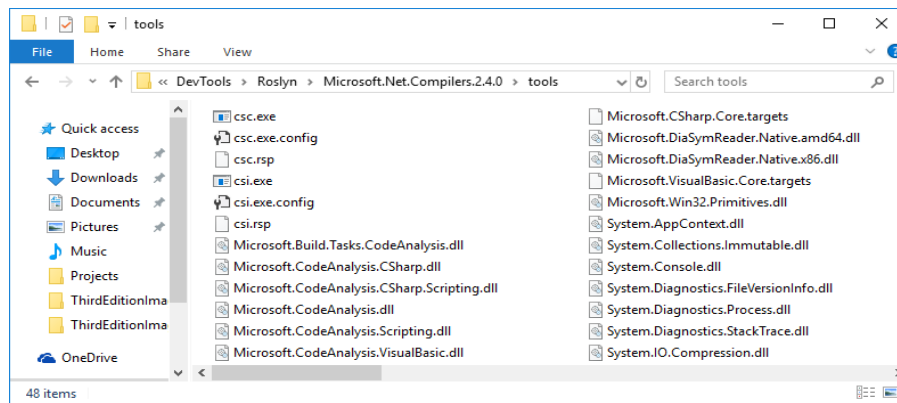


Figure 2-40: Contents of c:\DevTools\Roslyn\Microsoft.Net.Compilers.2.4.0\tools

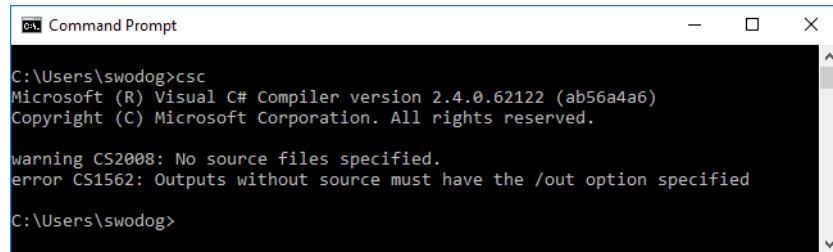
CONFIGURE ENVIRONMENT VARIABLES

Using the environment variable configuration process described earlier in this chapter, I recommend creating a new user environment variable named `ROSLYN_COMPILER_HOME` and setting its value to the path to the Roslyn compiler tools directory.

Next, and this is the tricky part, add the `ROSLYN_COMPILER_HOME` environment variable to the user `PATH` environment variable, and make sure it appears above or before the `DOT_NET_FRAMEWORK_HOME` environment variable. The reason for this ordering is that Windows will execute the first `csc.exe` it finds in the `PATH`. (**Note:** You could actually remove the `DOT_NET_FRAMEWORK_HOME` environment variable from the `PATH` if you use the Roslyn compiler.)

Compile and Run a Test Program

When you've finished twiddling with the environment variables do a test drive. Open a command prompt (**not** the Visual Studio Developer Command Prompt) and type `csc`. Your output should look similar to figure 2-41.



```
cs Command Prompt
C:\Users\swodog>csc
Microsoft (R) Visual C# Compiler version 2.4.0.62122 (ab56a4a6)
Copyright (C) Microsoft Corporation. All rights reserved.

warning CS2008: No source files specified.
error CS1562: Outputs without source must have the /out option specified

C:\Users\swodog>
```

Figure 2-41: Testing Roslyn C# Compiler

Quick Review

To compile C# programs that use features found in language versions 6 and above you'll need to use the Roslyn C# compiler. If you installed Visual Studio you already have access to the Roslyn compiler from the command-line via the Visual Studio Developer Command Prompt.

To install the Roslyn compilers stand alone, first download and install the NuGet package manager, then use it to download and install the Microsoft.Net.Compilers package. Set your environment variables to point to the Roslyn compiler tools directory. Pay special attention to PATH environment variable ordering to ensure you are calling the correct C# compiler.

LINQPad

Another great tool available to help you learn C# is LINQPad. LINQPad lets you quickly test C#, F#, and Visual Basic code, and connect to and query databases using LINQ (Language Integrated Query) or SQL. You can obtain the standard edition of LINQPad free of charge.

Installation is straightforward. I recommend you check the box to add `lprun.exe` to the PATH to let you run LINQPad from the command-line as is shown in Figure 2-42.

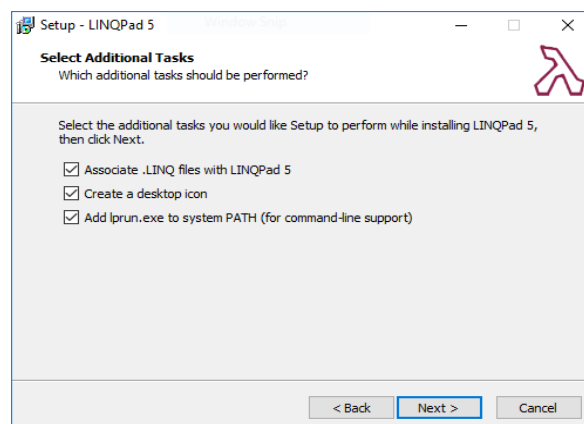


Figure 2-42: Check the Box to Add `lprun.exe` to the PATH

Figure 2-43 shows a LINQPad session in progress running HelloWorld.

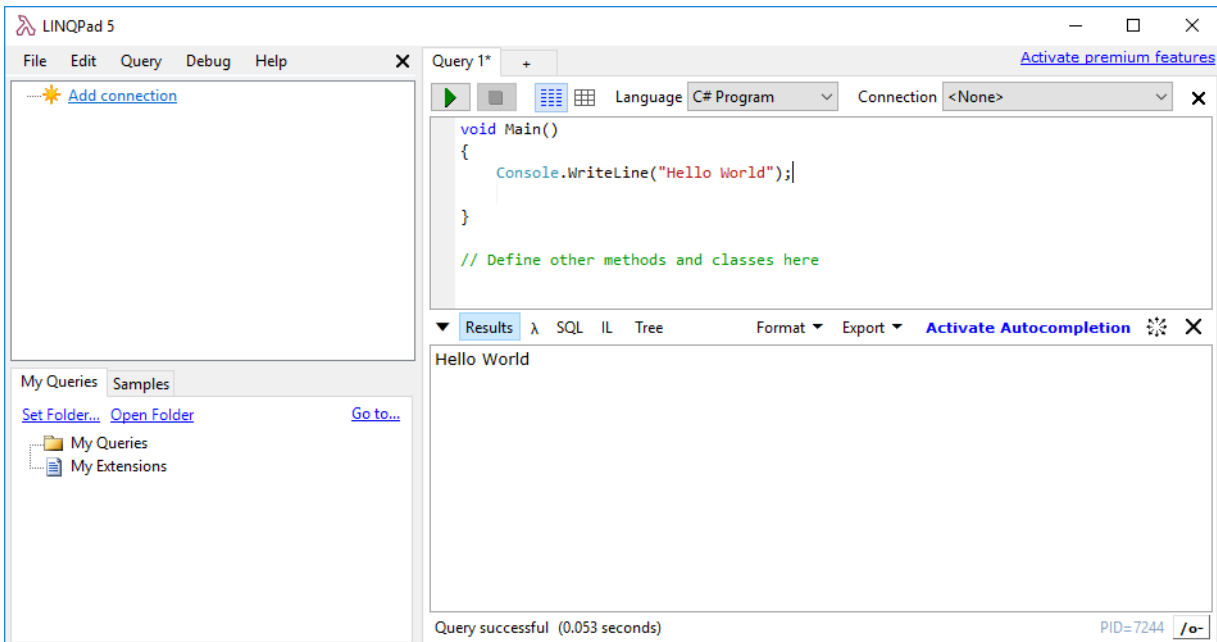


Figure 2-43: Running HelloWorld in LINQPad

Referring to figure 2-43 — Note how the code only contains the Main() method and the output of the program is displayed below the code in the Results window. Later in the book I'll show you how to use LINQPad to connect to and query a Microsoft SQL Server database using LINQ and SQL.

Quick Review

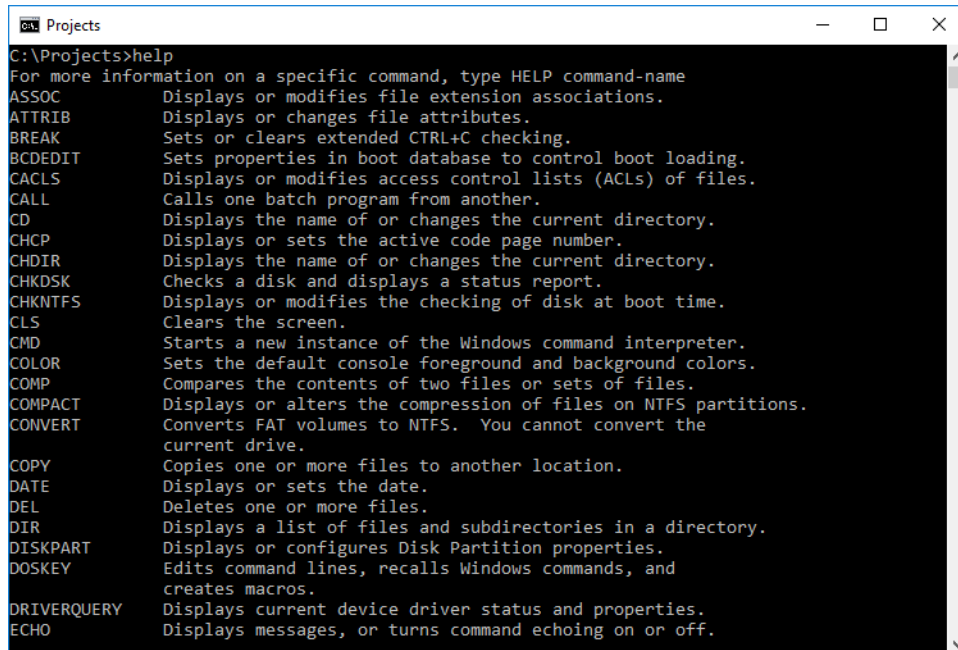
LINQPad is a great tool for quickly testing short C# code snippets or programs. I recommend downloading LINQPad and using it as a study aid.

Helpful Command-Prompt Commands

In this section, I want to provide you with a handful of command-prompt commands you'll find helpful as you start down the path of Windows software development. You may wonder, "Why bother with the command-prompt when you can just point, click, select, and drag things with a mouse?" Quite simply, a little knowledge of the command-prompt lets you work faster and more efficiently. Let's start with the help command.

help (GETTING HELP ON COMMAND-PROMPT COMMANDS)

Type `help` at the command prompt to get a listing of commands as is shown in figure 2-44. To get help for a specific command type `help` followed by the command. For example, to get help on the `cd` command type `help cd` as shown in figure 2-45.

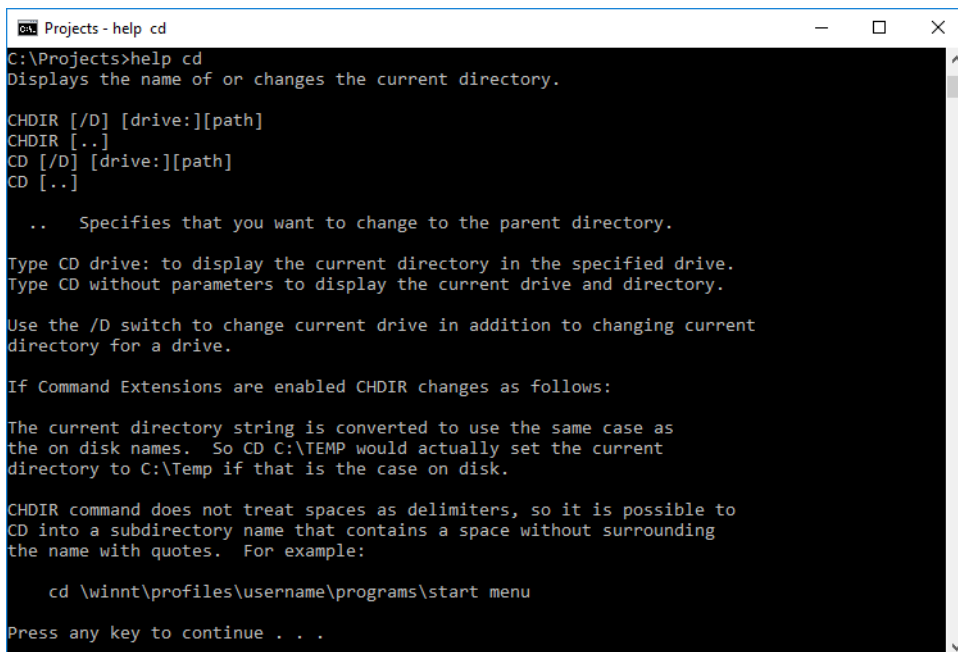


```

C:\Projects>help
For more information on a specific command, type HELP command-name
ASSOC      Displays or modifies file extension associations.
ATTRIB    Displays or changes file attributes.
BREAK     Sets or clears extended CTRL+C checking.
BCDEDIT   Sets properties in boot database to control boot loading.
CACLS     Displays or modifies access control lists (ACLs) of files.
CALL      Calls one batch program from another.
CD        Displays the name of or changes the current directory.
CHCP     Displays or sets the active code page number.
CHDIR    Displays the name of or changes the current directory.
CHKDSK   Checks a disk and displays a status report.
CHKNTFS  Displays or modifies the checking of disk at boot time.
CLS      Clears the screen.
CMD      Starts a new instance of the Windows command interpreter.
COLOR    Sets the default console foreground and background colors.
COMP     Compares the contents of two files or sets of files.
COMPACT  Displays or alters the compression of files on NTFS partitions.
CONVERT  Converts FAT volumes to NTFS. You cannot convert the
         current drive.
COPY     Copies one or more files to another location.
DATE     Displays or sets the date.
DEL      Deletes one or more files.
DIR      Displays a list of files and subdirectories in a directory.
DISKPART Displays or configures Disk Partition properties.
DOSKEY   Edits command lines, recalls Windows commands, and
         creates macros.
DRIVERQUERY Displays current device driver status and properties.
ECHO     Displays messages, or turns command echoing on or off.

```

Figure 2-44: help Command Partial Listing



```

C:\Projects>help cd
Displays the name of or changes the current directory.

CHDIR [/D] [drive:][path]
CHDIR [..]
CD [/D] [drive:][path]
CD [..]

  .. Specifies that you want to change to the parent directory.

Type CD drive: to display the current directory in the specified drive.
Type CD without parameters to display the current drive and directory.

Use the /D switch to change current drive in addition to changing current
directory for a drive.

If Command Extensions are enabled CHDIR changes as follows:

The current directory string is converted to use the same case as
the on disk names. So CD C:\TEMP would actually set the current
directory to C:\Temp if that is the case on disk.

CHDIR command does not treat spaces as delimiters, so it is possible to
CD into a subdirectory name that contains a space without surrounding
the name with quotes. For example:

    cd \winnt\profiles\username\programs\start menu

Press any key to continue . . .

```

Figure 2-45: Getting help on the cd Command

dir (List dIRECTORY CONTENTS)

Use the `dir` command to get a listing of directory contents. The `dir` command has numerous options which you can explore more deeply by typing `help dir`, but I use it mostly just as-is. Just type `dir` to get a listing of the current directory. To list a subdirectory in the current directory type `dir` then a space and

then the name of the directory. You can use absolute paths as well to list the contents of any directory from any other directory, but I find this tedious. Figure 2-46 shows the `dir` command being run in my `c:\Projects\BookProjects` directory.

```

Volume Serial Number is DC59-2979

Directory of C:\Projects\BookProjects

12/02/17  10:27 AM  <DIR>      .
12/02/17  10:27 AM  <DIR>      ..
12/02/17  10:27 AM  <DIR>      Chapter10
12/02/17  10:27 AM  <DIR>      Chapter11
12/02/17  10:27 AM  <DIR>      Chapter12
12/02/17  10:27 AM  <DIR>      Chapter13
12/02/17  10:27 AM  <DIR>      Chapter14
12/02/17  10:27 AM  <DIR>      Chapter15
12/02/17  10:27 AM  <DIR>      Chapter16
12/02/17  10:27 AM  <DIR>      Chapter17
12/02/17  10:27 AM  <DIR>      Chapter19
12/26/17  03:14 PM  <DIR>      Chapter2
12/02/17  10:27 AM  <DIR>      Chapter20
12/02/17  10:27 AM  <DIR>      Chapter21
12/02/17  10:27 AM  <DIR>      Chapter22
12/02/17  10:27 AM  <DIR>      Chapter23
12/02/17  10:27 AM  <DIR>      Chapter24
12/02/17  10:27 AM  <DIR>      Chapter25
12/02/17  10:53 AM  <DIR>      Chapter3
12/26/17  11:55 AM  <DIR>      Chapter4
12/02/17  10:27 AM  <DIR>      Chapter6
12/02/17  10:27 AM  <DIR>      Chapter7
03/03/18  09:24 AM  <DIR>      Chapter8
12/02/17  10:27 AM  <DIR>      Chapter9
12/02/17  10:27 AM  <DIR>      EmployeePics
                0 File(s)          0 bytes
                25 Dir(s)  210,219,520,000 bytes free

C:\Projects\BookProjects>

```

Figure 2-46: `dir` Command

TREE (Display GRAPHIC DIRECTORY STRUCTURE)

The `tree` command gives you a graphical display of a directory's contents as is shown in figure 2-47.

```

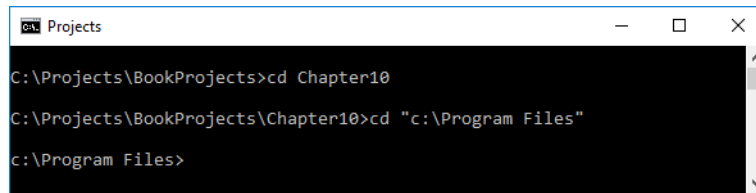
Chapter2
├── VisualStudio
│   └── HelloWorld
│       └── HelloWorld
│           ├── bin
│           │   ├── Debug
│           │   └── Release
│           ├── obj
│           │   └── x86
│           │       ├── Debug
│           │       │   ├── TempPE
│           │       │   └── Release
│           │       └── TempPE
│           └── Properties
└── Chapter20
    ├── EmployeeTrainingApp
    │   └── FifthIteration
    │       ├── Client
    │       │   ├── app
    │       │   ├── build
    │       │   ├── config
    │       │   └── ref
    │       └── Server

```

Figure 2-47: `tree` Command Partial Listing

cd (CHANGE DIRECTORY)

Use the `cd` command to change directories. Some terms are in order here. The working directory is the directory you're currently in. To change to a subdirectory simply type `cd` and the name of the subdirectory. To change from the current or working directory to the parent directory use the shortcut `cd ..` — that's the command `cd` followed by two dots. You'll need to surround the path with double quotes if any of the directories in the path contain a space as shown in figure 2-48.



```
C:\Projects\BookProjects>cd Chapter10
C:\Projects\BookProjects\Chapter10>cd "c:\Program Files"
c:\Program Files>
```

Figure 2-48: `cd` Command — Paths with Spaces Require Double Quotes

copy AND xcopy (COPY files AND DIRECTORIES)

Use the `copy` command to copy files from one location to another. Use the `xcopy` command to copy files and entire directory structures. These commands have complex options which I encourage you to lookup using the command-prompt `help` command.

md AND mkdir (MAKE DIRECTORY)

The `md` and `mkdir` commands are synonymous. They are both used to create new directories. Simply type either `md` or `mkdir` followed by the name of the directory. You can use absolute path names as well.

del AND erase (DELETE files)

The `del` and `erase` commands are synonymous and are used to delete files. Be careful with these commands especially if you use wild card characters. For example, to delete every file in a directory that ends with `.txt` you would type `erase *.txt`.

rd AND rmdir (REMOVE DIRECTORY)

The `rd` and `rmdir` commands are synonymous and are used to delete entire directories. Use with caution!

WHERE TO GO FROM HERE

To learn more about the command-prompt including how to combine commands to create batch files I recommend Elias Bachaalany's excellent book "batchography: The Art of Batch Files Programming".

Quick REVIEW

Knowledge of how to effectively use the command-prompt will make you a more efficient programmer and can potentially save you a lot of time fiddling with windows and folders.

MICROSOFT IMAGINE

Microsoft Imagine, <http://imagine.microsoft.com>, allows registered students and faculty to gain access to professional development tools and other software. You already have access to some pretty powerful development tools via Microsoft's Visual Studio website, free of charge. Schools can subscribe to Microsoft Imagine and make other development tools available as well, like Windows Server operating systems.

Check with your school to find out if they subscribe to Microsoft Imagine and how to access the subscription. You may be surprised at the amount of additional software you have at your disposal.

SUMMARY

All you need to create robust Microsoft C# applications is a good text editor and the C# compiler that's included with the .NET Framework. Both can be obtained free of charge.

You must configure your development environment before you can compile programs from the command-line. This includes creating or editing one or more operating system environment variables. An environment variable is used to store data about the operating system environment. There are generally two types of environment variables: *system variables* and *user variables*. System environment variables store data that affects the operating system environment for all users; user environment variables store data that affects the operating system environment for a particular user.

Environment variable values can be accessed by enclosing the variable name in percent '%' characters.

The operating system uses the PATH environment variable to locate executable files. You must create or edit the PATH environment variable to include the full path to the C# compiler (csc.exe).

It's helpful to create a project folder and a shortcut to the command prompt on your desktop. Set the command prompt shortcut's **Start in** property so it will automatically open in your designated projects folder. Increase the command prompt shortcut's screen buffer **height** and **width** properties to see more information in the console window.

It's also a good idea to set your folder options to display file type extensions. This will prevent headaches associated with accidentally saving source files with a ".txt" extension.

To create a C# program you must first create the source file, compile the source file with the `csc` compiler tool, and then execute the program by typing its name at the command prompt and pressing the Return or Enter key.

You're bound to get a few compiler errors when you start writing your own programs. Use Google to search for the error code. This will lead you straight to the answer on Microsoft's documentation website. Remember to always **fix the first compiler error first!**

Visual Studio is a powerful Integrated Development Environment (IDE) that combines editing, compiling, debugging, code profiling, and other services. Visual Studio Community is freely available from Microsoft.

Even though Visual Studio is a powerful tool, it will serve you well to learn how to compile C# programs from the command-line. This will enable you to better understand what's happening under the covers and let you fix things when something goes wrong.

If you install Visual Studio the Roslyn compilers are already installed. To use the Roslyn compiler from the command-line simply open the Visual Studio Developer Command Prompt.

To compile C# programs that use features found in language versions 6 and above you'll need to use the Roslyn C# compiler. If you installed Visual Studio you already have access to the Roslyn compiler from the command-line via the Visual Studio Developer Command Prompt.

To install the Roslyn compilers stand alone, first download and install the NuGet package manager, then use it to download and install the Microsoft.Net.Compilers package. Set your environment variables to point to the Roslyn compiler tools directory. Pay special attention to PATH environment variable ordering to ensure you are calling the correct C# compiler.

LINQPad is a great tool for quickly testing short C# code snippets or programs. I recommend downloading LINQPad and using it as a study aid.

If you're a registered student or faculty member, check with your institution to see if they subscribe to Microsoft Imagine. You may have access to more powerful software development tools.

Skill-Building Exercises

1. **Creating and Using Environment Variables:** Create an environment variable named "PROJECTS_HOME". For its value use the path to your projects folder.
2. **Setting Up Your Development Environment:** Set up your development environment following the steps outlined in this chapter. Test your development environment by compiling and running the program given in example 2.1.
3. **Web Research:** Visit Microsoft documentation website <https://docs.microsoft.com> and familiarize yourself with the information it contains. Locate the C# compiler errors page and bookmark the page in your web browser.
4. **Roslyn Compiler Installation:** Download and install the Roslyn compiler using the process explained in this chapter. If you've already configured environment variables to compile programs from the command-line using the .NET Framework, be sure to order the ROSLYN_COMPILER_HOME environment variable correctly in the PATH to ensure the correct version of the C# compiler is called. Optionally, you can remove the DOT_NET_FRAMEWORK_HOME environment variable from the PATH altogether.

Suggested Projects

1. **Alternative .NET Development Environments:** Microsoft produces versions of Visual Studio that run on Linux and Apple's OS X (macOS). If you're a Linux or Mac user, download Visual Studio for your particular environment and create the HelloWorld project discussed in this chapter.
2. **Using LINQPad:** Download and install LINQPad. Use it to run the HelloWorld code listed in Example 2.2. but change the Console.ReadKey() method to Console.Read().

Self-Test Questions

1. What two things, at minimum, do you need to do C#.NET development?
2. What is an operating system environment variable?

3. What is the difference between a user environment variable vs. a system environment variable?
4. List the general steps required to create environment variables in Windows 7 and 10.
5. What characters must you use before and after an environment variable to get its value?
6. What is the purpose of the PATH environment variable?
7. What should you do if you get more than one compiler error?
8. What's the advantage of using an IDE like Visual Studio?
9. What are the general steps required to create, compile, and execute a C# program?
10. List a couple of steps you might take to better help you see the output of a console application that runs briefly and terminates immediately?

REFERENCES

Microsoft Developer Network website: <https://www.msdn.com> or <https://developer.microsoft.com>

Microsoft Visual Studio website: <https://www.visualstudio.com>

Microsoft TechNet website: <https://technet.microsoft.com>

Microsoft Docs website: <https://docs.microsoft.com>

NOTES
