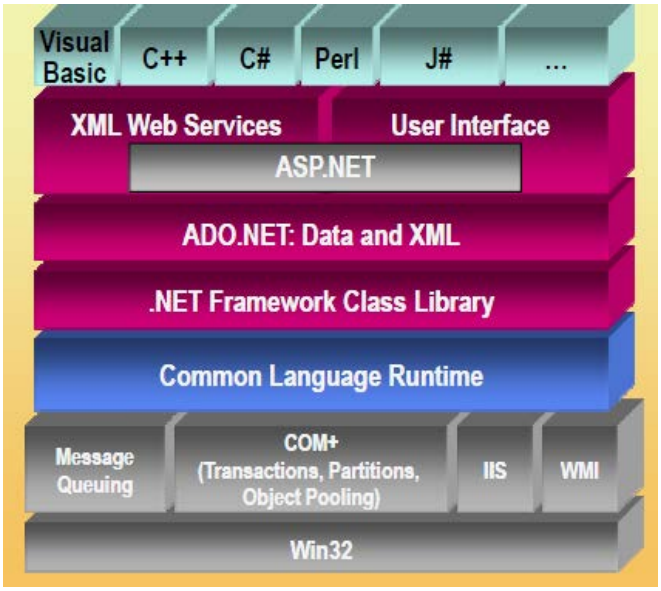
Introduction to C#
Instructor: Frank Stepanski

What's in the .NET Framework?

The .NET Framework consists primarily of a gigantic library of code that you use from your client languages (such as C#) using object-oriented programming (OOP) techniques.

This library is categorized into different modules — you use portions of it depending on the results you want to achieve.

For example, one module contains the building blocks for Windows applications, another for network programming, and another for web development. Some modules are divided into more specific sub-modules, such as a module for building web services within the module for web development.



The intention is for different operating systems to support some or all of these modules, depending on their characteristics.

A smart phone, for example, includes support for all the core .NET functionality but is unlikely to require some of the more esoteric modules.

Part of the .NET Framework library defines some basic *types*.

A type is a representation of data, and specifying some of the most fundamental of these (such as "a 32-bit signed integer") facilitates interoperability between languages using the .NET Framework. This is called the *Common Type System* (CTS).

As well as supplying this library, the .Net Framework also includes the .NET *Common Language Runtime* (CLR), which is responsible for the execution of all applications developed using the .NET library.

Writing Applications Using the .NET Framework

Writing an application using the .NET Framework means writing code (using any of the languages that support the Framework) using the .NET code library.

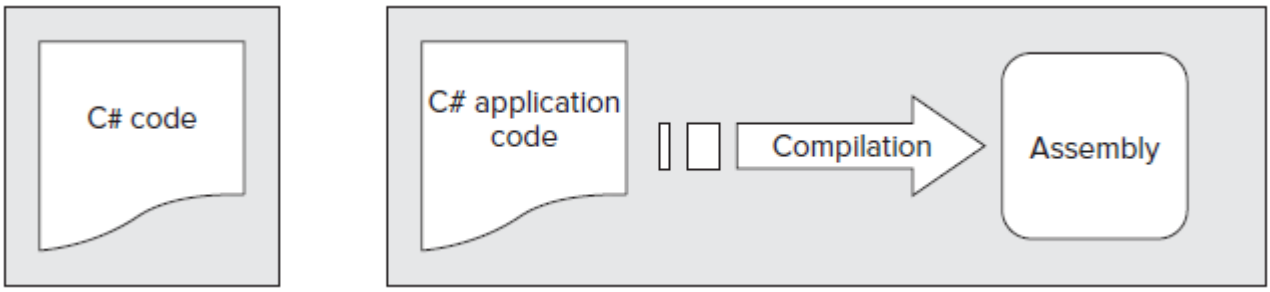
In this class you will use Visual Studio for your development. VS is a powerful, integrated development environment that supports C# (as well as managed and unmanaged C++, Visual Basic, and some others).

Note: We will use the free version of Visual Studio which focuses on desktop development (Express 2013 for Windows).

The advantage of this environment is the ease with which .NET features can be integrated into your code. The code that you create will be entirely C# but use the .NET Framework throughout the project.

In order for C# code to execute, it must be converted into a language that the target operating system understands, known as *native code*. This conversion is called *compiling* code, an act that is performed by a *compiler*. Under the .NET Framework, this is a two-stage process, CIL and JIT.

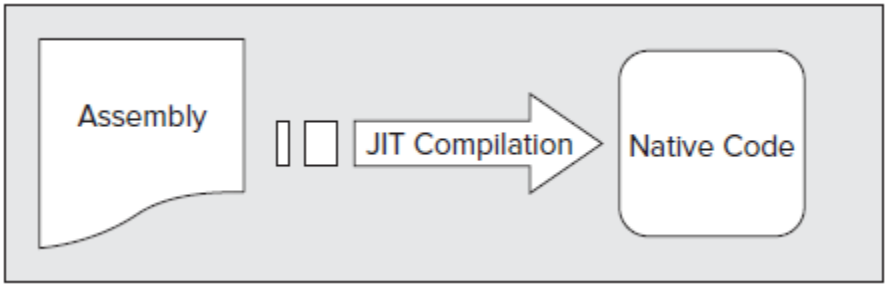
When you compile code that uses the .NET Framework library, you don't immediately create operating system–specific native code. Instead, you compile your code into *Common Intermediate Language* (CIL) code.



This code isn't specific to any operating system (OS) and isn't specific to C#. Other .NET languages — Visual Basic .NET, for example — also compile to this language as a first stage.

This compilation step is carried out by VS when you develop C# applications. Obviously, more work is necessary to execute an application.

That is the job of a *just-in-time* (JIT) compiler, which compiles CIL into native code that is specific to the OS and machine architecture being targeted.



Only at this point can the OS execute the application. The *just-in-time* part of the name reflects the fact that CIL code is compiled only when it is needed. This compilation can happen on the fly while your application is running, although luckily this isn't something that you normally need to worry about as a developer.

Assemblies

When you compile an application, the CIL code is stored in an *assembly*. Assemblies include both executable application files that you can run directly from Windows without the need for any other programs (these have a .exe file extension) and libraries (which have a .dll extension) for use by other applications.

In addition to containing CIL, assemblies also include *meta* information (that is, information about the information contained in the assembly, also known as *metadata*) and optional *resources* (additional data used by the CIL, such as sound files and pictures).

The meta information enables assemblies to be fully self-descriptive. You need no other information to use an assembly, meaning you avoid situations such as failing to add required data to the system registry and so on, which was often a problem when developing with other platforms.

This means that deploying applications is often as simple as copying the files into a directory on a remote computer. Because no additional information is required on the target systems, you can just run an executable file from this directory and (assuming the .NET CLR is installed) you're good to go.

Managed Code

The role of the CLR doesn't end after you have compiled your code to CIL and a JIT compiler has compiled that to native code. Code written using the .NET Framework is *managed* when it is executed (a stage usually referred to as *runtime*).

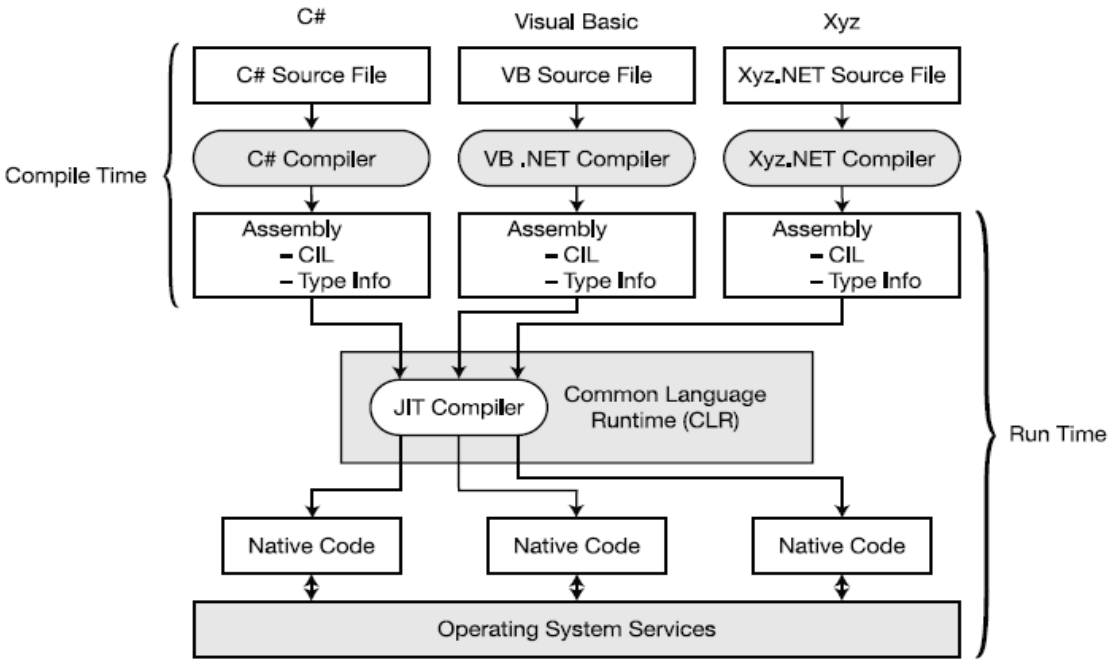
This means that the CLR looks after your applications by managing memory, handling security, allowing cross-language debugging, and so on.

Garbage Collection

One of the most important features of managed code is the concept of *garbage collection*. This is the .NET method of making sure that the memory used by an application is freed up completely when the application is no longer in use.

Prior to .NET this was mostly the responsibility of programmers, and a few simple errors in the code could result in large blocks of memory mysteriously disappearing as a result of being allocated to the wrong place in memory. That usually meant a progressive slowdown of your computer, followed by a system crash.

.NET garbage collection works by periodically inspecting the memory of your computer and removing anything from it that is no longer needed. There is no set time frame for this; it might happen thousands of times a second, once every few seconds, or whenever, but you can rest assured that it will happen.



Introducing C#

C#, as mentioned earlier, is one of the languages you can use to create applications that will run in the .NET CLR.

It is an evolution of the C and C++ languages and has been created by Microsoft specifically to work with the .NET platform. The C# language has been designed to incorporate many of the best features from other languages, while clearing up their problems.

C# uses the framework and therefore has no restrictions on possible applications. However, here are a few of the more common application types:

- **Desktop applications** — Applications, such as Microsoft Office, that have a familiar Windows look and feel about them. This is made simple by using the Windows Presentation Foundation (WPF) module of the .NET Framework, which is a library of *controls* (such as buttons, toolbars, menus, and so on) that you can

use to build a Windows user interface (UI).

- **Windows Store applications** — Windows 8 has introduced a new type of application, known as a Windows Store application. This type of application is designed primarily for touch devices, and it is usually run full-screen, with a minimum of clutter, and an emphasis on simplicity. You can create these applications in several ways, including using WPF.

- **Web applications** — Web pages such as those that might be viewed through any web browser. The .NET Framework includes a powerful system for generating web content dynamically, enabling personalization, security, and much more. This system is called ASP.NET (Active Server Pages .NET), and you can use C# to create ASP.NET applications using Web Forms.

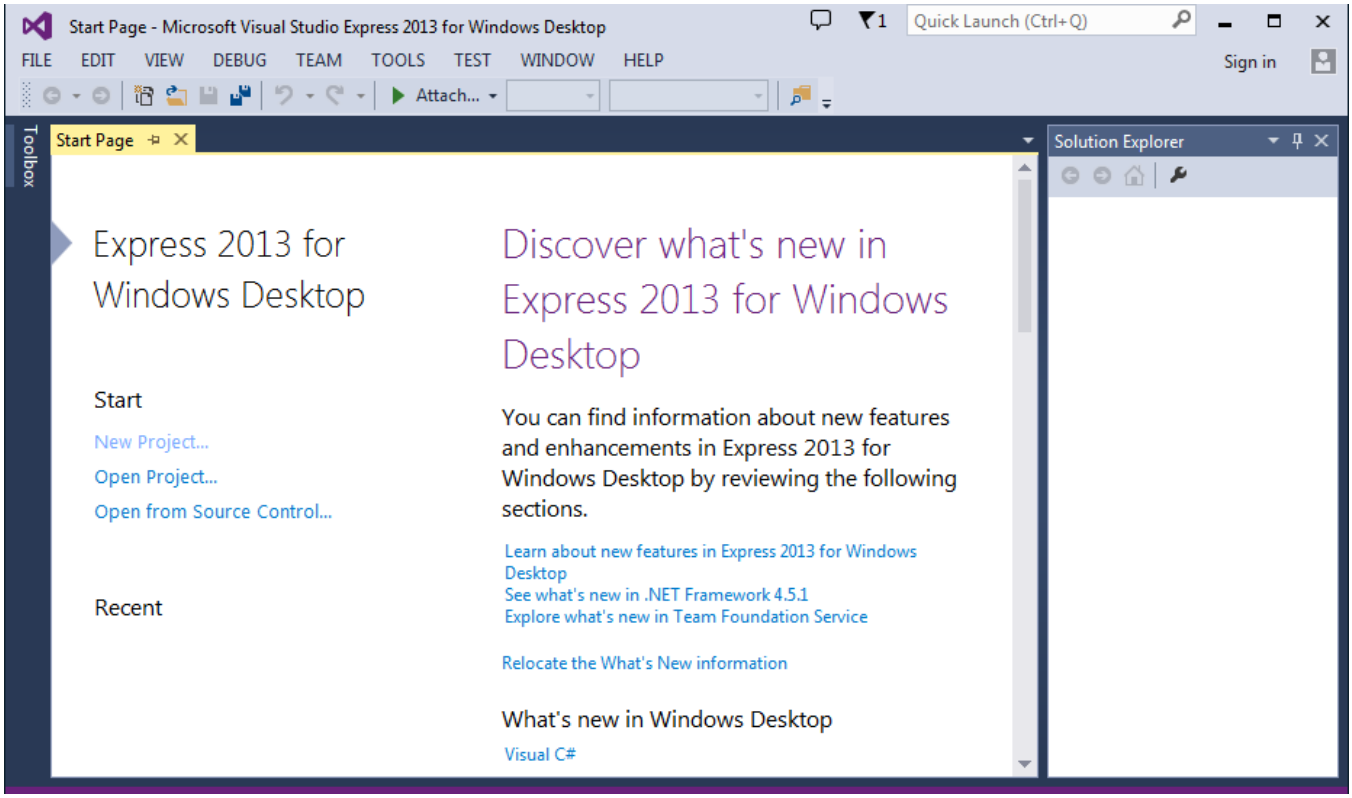
- **WCF services** — A way to create versatile distributed applications. Using WCF you can exchange virtually any data over local networks or the Internet, using the same simple syntax regardless of the language used to create a service or the system on which it resides.

Writing Your First C# Program

When VS is first loaded, it immediately presents you with a host of windows, most of which are empty, along with an array of menu items and toolbar icons. You will only be using a very small portion of these in this course.

The main window, which contains a helpful Start Page by default when VS is started, is where all your code is displayed. This window can contain many documents, each indicated by a tab, so you can easily switch between several files by clicking their filenames.

Above the main window are toolbars and the VS menu. Several different toolbars can be placed here, with functionality ranging from saving and loading files to building and running projects to debugging controls.

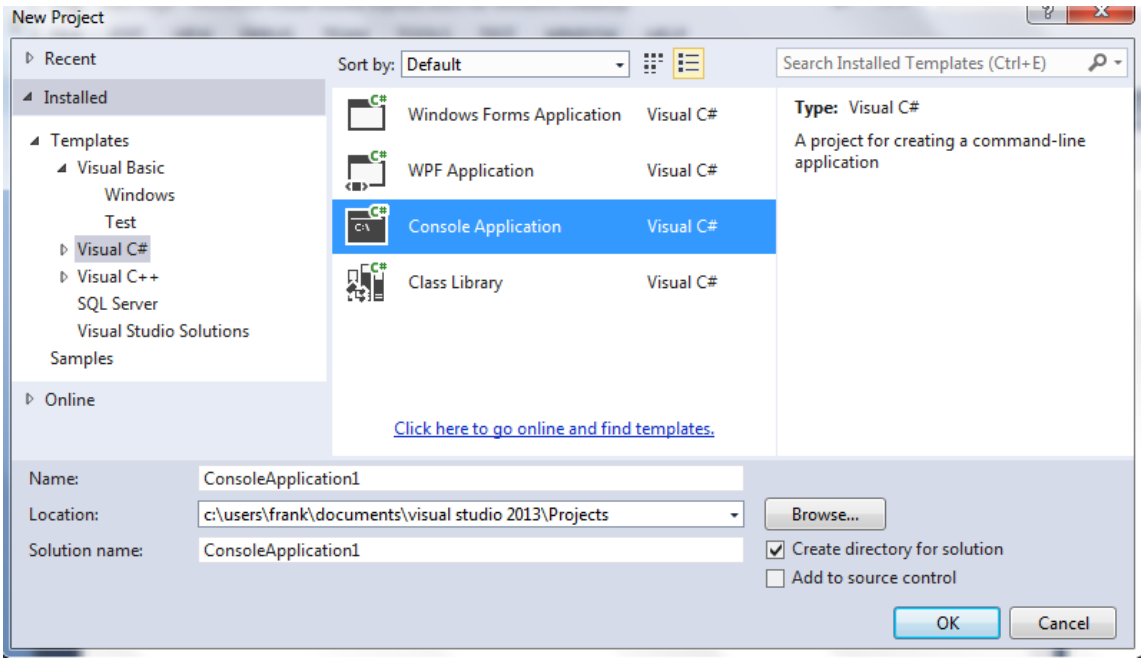


Console Applications

You will use console applications regularly in this class because it is the best way to focus on the concepts of the C# language and not on the many aspects of the .NET framework.

Create a new console application project by:

1. Selecting File ➪ New ➪ Project



2. Ensure that the Visual C# node is selected in the left pane and choose Console Application project type in the middle pane.

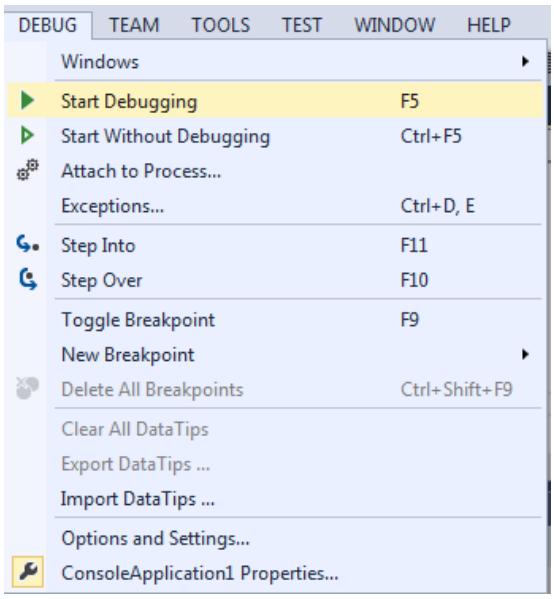
3. Once the project is initialized, add the following lines of code to the file displayed in the main window:

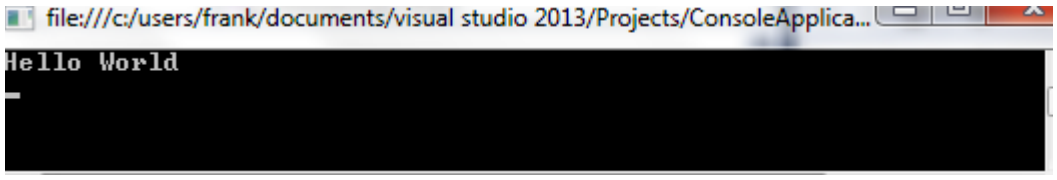
```
namespace ConsoleApplication1
{

class Program
     {
          static void Main(string[] args)
          {
               // Output text to the screen.
               Console.WriteLine("Hello World");
               Console.ReadKey();
          }

     }
}
```

4. Select the Debug ➪ Start Debugging menu item. After a few moments you should see the window.





5. Press any key to exit the application (you might need to click on the console window to focus on it first).

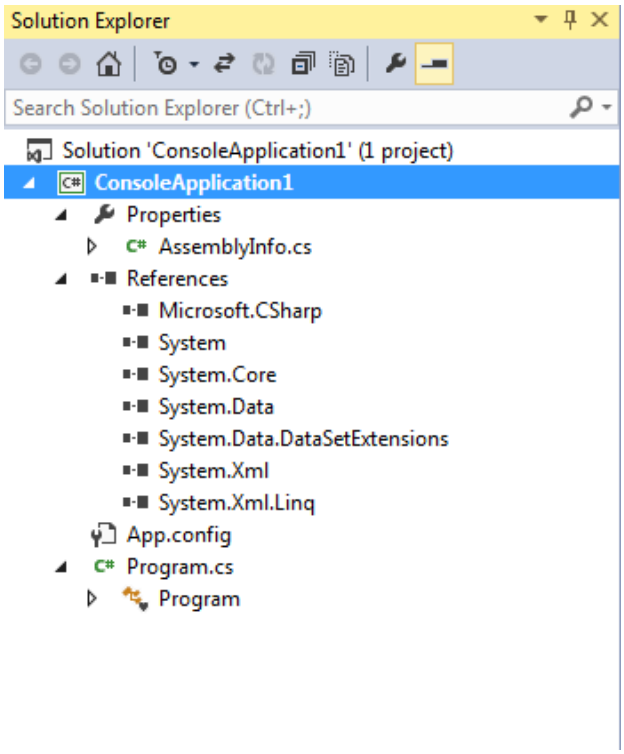
Your code can be compiled and executed in several ways.

The process you used in the example — selecting Debug ➪ Start Debugging — also has a keyboard shortcut (F5) and a toolbar icon.

You can also run code without being in debugging mode using the Debug ➪ Start Without Debugging menu item (or by pressing Ctrl+F5), or compile your project without running it (with debugging on or off) using Build ➪ Build Solution or pressing F6.

The Solution Explorer

By default, the Solution Explorer window is docked in the top-right corner of the screen. As with other windows, you can move it wherever you like, or you can set it to auto-hide by clicking the pin icon.



The Solution Explorer window shares space with another useful window called Class View, which you can display using View ➪ Class View.

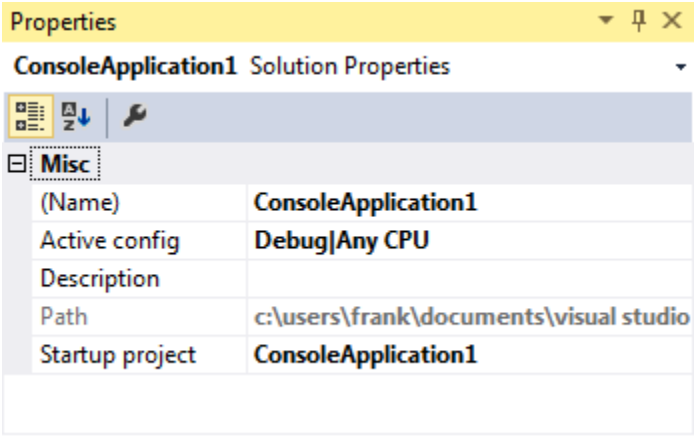
This Solution Explorer view shows the files that make up the ConsoleApplication1 project.

The file to which you added code, Program.cs, is shown along with another code file, AssemblyInfo.cs, and several references

You can use this window to change what code is displayed in the main window by double-clicking .cs files; right-clicking them and selecting View Code; or by selecting them and clicking the toolbar button that appears at the top of the window.

You can expand code items such as **Program.cs** to see what is contained. This overview of your code structure can be a very useful tool; it also enables you to navigate directly to specific parts of your code file, instead of opening the code file and scrolling to the part you want.

The Properties Window

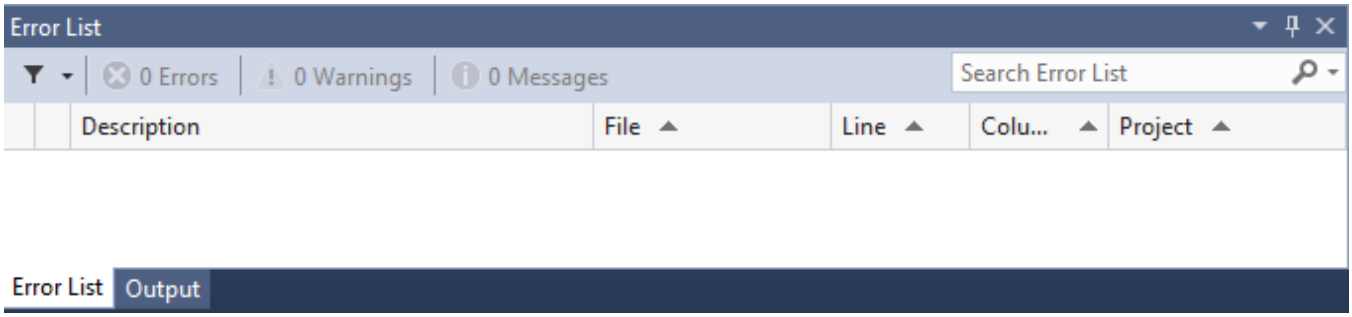


The Properties window (select View ➪ Properties Window if it isn't already displayed) shows additional information about whatever you select in the window above it.

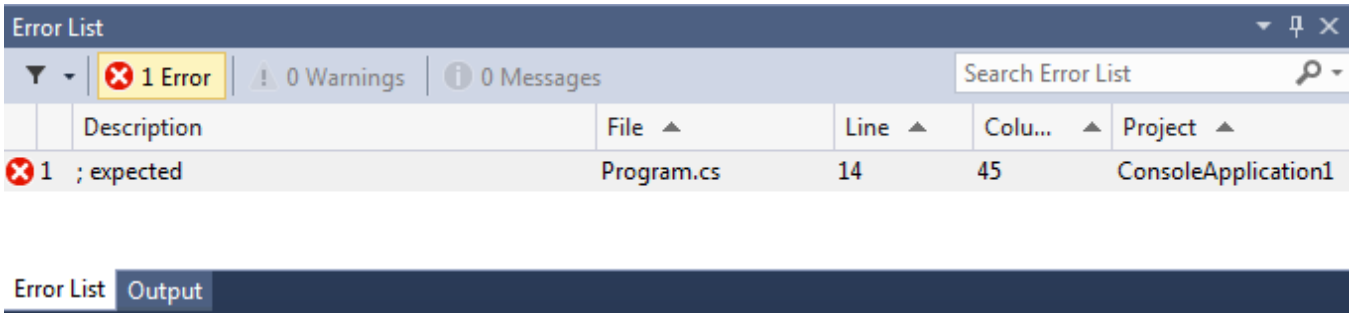
This window also displays information about other selected items, such as user interface components.

The Error List Window

Currently, the Error List window (View ➪ Error List) isn't showing anything interesting because there is nothing wrong with the application.



As a test, remove the semicolon from one of the lines of code you added in the previous section.



This window helps you eradicate bugs in your code because it keeps track of what you have to do to compile projects. If you double-click the error shown here, the cursor jumps to the position of the error in your source code (the source file containing the error will be opened if it isn't already open), so you can fix it quickly.

Red wavy lines appear at the positions of errors in the code, so you can quickly scan the source code to see where problems lie.