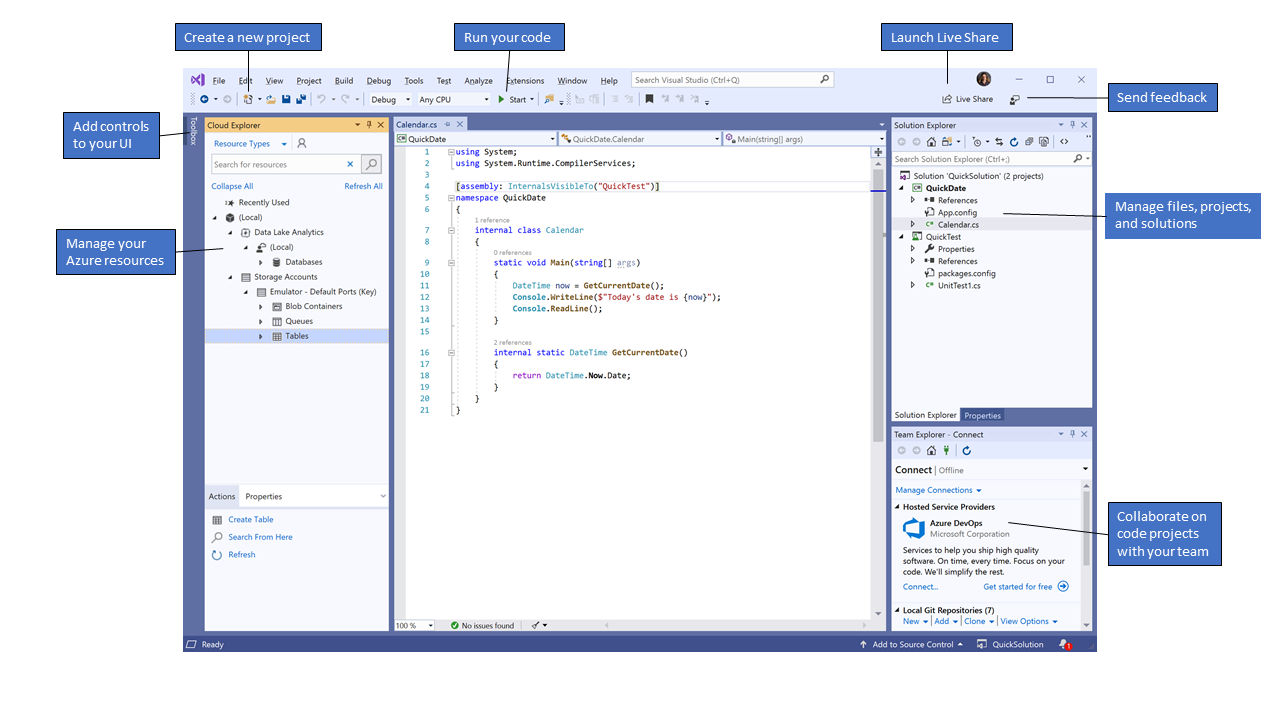
**Welcome to the Visual Studio IDE**

* 14 minutes to read

The Visual Studio *integrated development environment* is a creative launching pad that you can use to edit, debug, and build code, and then publish an app. An integrated development environment (IDE) is a feature-rich program that can be used for many aspects of software development. Over and above the standard editor and debugger that most IDEs provide, Visual Studio includes compilers, code completion tools, graphical designers, and many more features to ease the software development process.

[](https://docs.microsoft.com/en-us/visualstudio/get-started/media/vs-2019/ide-overview.png?view=vs-2019#lightbox)

This image shows Visual Studio with an open project and several key tool windows you'll likely use:

* [Solution Explorer](https://docs.microsoft.com/en-us/visualstudio/ide/solutions-and-projects-in-visual-studio?view=vs-2019) (top right) lets you view, navigate, and manage your code files. **Solution Explorer** can help organize your code by grouping the files into [solutions and projects](https://docs.microsoft.com/en-us/visualstudio/get-started/tutorial-projects-solutions?view=vs-2019).
* The [editor window](https://docs.microsoft.com/en-us/visualstudio/ide/writing-code-in-the-code-and-text-editor?view=vs-2019) (center), where you'll likely spend a majority of your time, displays file contents. This is where you can edit code or design a user interface such as a window with buttons and text boxes.
* [Team Explorer](https://docs.microsoft.com/en-us/azure/devops/user-guide/work-team-explorer) (bottom right) lets you track work items and share code with others using version control technologies such as [Git](https://git-scm.com/) and [Team Foundation Version Control (TFVC)](https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/overview).

**Editions**

Visual Studio is available for Windows and Mac. [Visual Studio for Mac](https://docs.microsoft.com/en-us/visualstudio/mac/) has many of the same features as Visual Studio 2019, and is optimized for developing cross-platform and mobile apps. This article focuses on the Windows version of Visual Studio 2019.

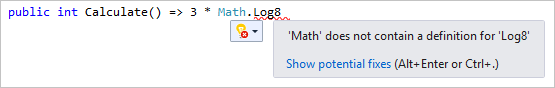
There are three editions of Visual Studio 2019: Community, Professional, and Enterprise. See [Compare Visual Studio editions](https://visualstudio.microsoft.com/vs/compare/) to learn about which features are supported in each edition.

**Popular productivity features**

Some of the popular features in Visual Studio that help you to be more productive as you develop software include:

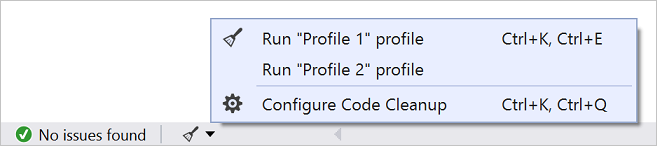
* Squiggles and [Quick Actions](https://docs.microsoft.com/en-us/visualstudio/ide/quick-actions?view=vs-2019)

Squiggles are wavy underlines that alert you to errors or potential problems in your code as you type. These visual clues enable you to fix problems immediately without waiting for the error to be discovered during build or when you run the program. If you hover over a squiggle, you see additional information about the error. A light bulb may also appear in the left margin with actions, known as Quick Actions, to fix the error.



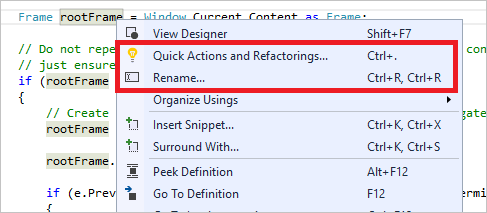
* Code Cleanup

With the click of a button, format your code and apply any code fixes suggested by your [code style settings](https://docs.microsoft.com/en-us/visualstudio/ide/reference/options-text-editor-csharp-formatting?view=vs-2019), [.editorconfig conventions](https://docs.microsoft.com/en-us/visualstudio/ide/create-portable-custom-editor-options?view=vs-2019), and [Roslyn analyzers](https://docs.microsoft.com/en-us/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2019). **Code Cleanup** helps you resolve issues in your code before it goes to code review. (Currently available for C# code only.)



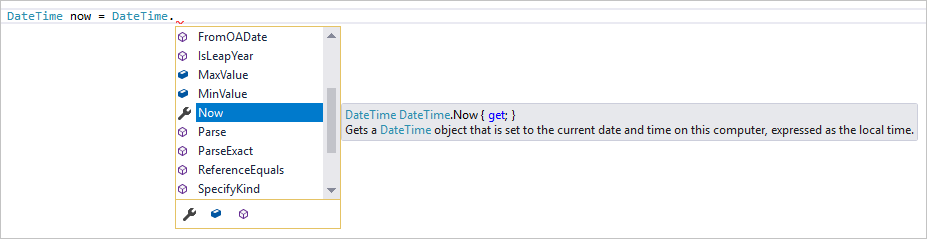
* [Refactoring](https://docs.microsoft.com/en-us/visualstudio/ide/refactoring-in-visual-studio?view=vs-2019)

Refactoring includes operations such as intelligent renaming of variables, extracting one or more lines of code into a new method, changing the order of method parameters, and more.



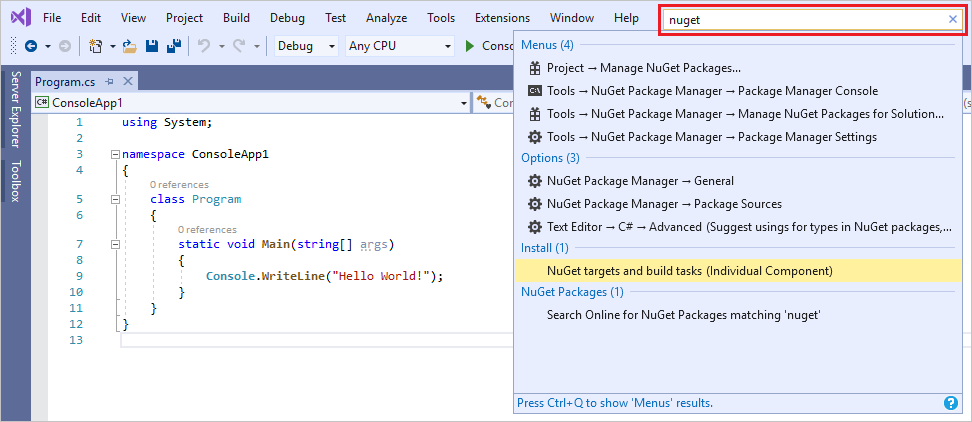
* [IntelliSense](https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2019)

IntelliSense is a term for a set of features that displays information about your code directly in the editor and, in some cases, write small bits of code for you. It's like having basic documentation inline in the editor, which saves you from having to look up type information elsewhere. IntelliSense features vary by language. For more information, see [C# IntelliSense](https://docs.microsoft.com/en-us/visualstudio/ide/visual-csharp-intellisense?view=vs-2019), [Visual C++ IntelliSense](https://docs.microsoft.com/en-us/visualstudio/ide/visual-cpp-intellisense?view=vs-2019), [JavaScript IntelliSense](https://docs.microsoft.com/en-us/visualstudio/ide/javascript-intellisense?view=vs-2019), and [Visual Basic IntelliSense](https://docs.microsoft.com/en-us/visualstudio/ide/visual-basic-specific-intellisense?view=vs-2019). The following illustration shows how IntelliSense displays a member list for a type:



* [Visual Studio search](https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-search?view=vs-2019)

Visual Studio can seem overwhelming at times with so many menus, options, and properties. Visual Studio search (**Ctrl**+**Q**) is a great way to rapidly find IDE features and code in one place.



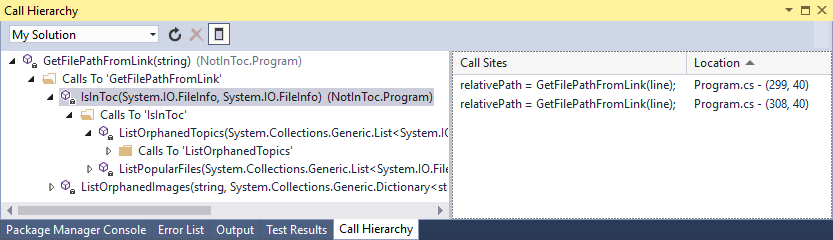
For information and productivity tips, see [How to use Visual Studio search](https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-search?view=vs-2019).

* [Live Share](https://docs.microsoft.com/en-us/visualstudio/liveshare/)

Collaboratively edit and debug with others in real time, regardless of what your app type or programming language. You can instantly and securely share your project and, as needed, debugging sessions, terminal instances, localhost web apps, voice calls, and more.

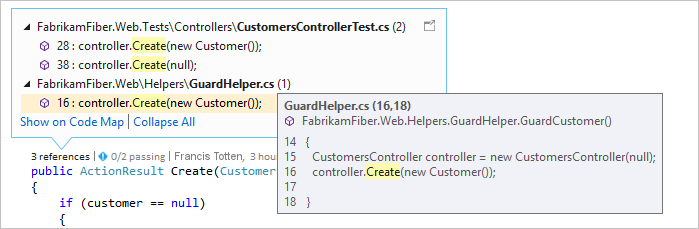
* [Call Hierarchy](https://docs.microsoft.com/en-us/visualstudio/ide/reference/call-hierarchy?view=vs-2019)

The **Call Hierarchy** window shows the methods that call a selected method. This can be useful information when you're thinking about changing or removing the method, or when you're trying to track down a bug.



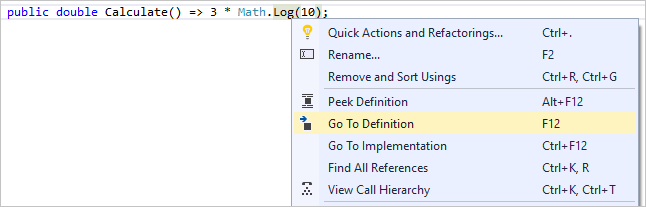
* [CodeLens](https://docs.microsoft.com/en-us/visualstudio/ide/find-code-changes-and-other-history-with-codelens?view=vs-2019)

CodeLens helps you find references to your code, changes to your code, linked bugs, work items, code reviews, and unit tests, all without leaving the editor.



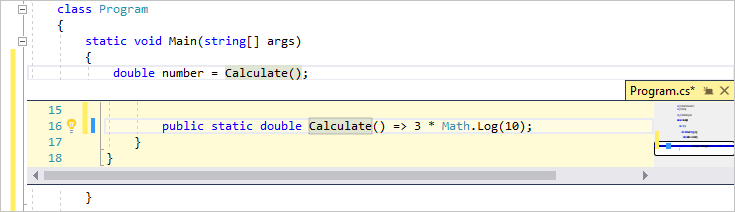
* [Go To Definition](https://docs.microsoft.com/en-us/visualstudio/ide/go-to-and-peek-definition?view=vs-2019)

The Go To Definition feature takes you directly to the location where a function or type is defined.



* [Peek Definition](https://docs.microsoft.com/en-us/visualstudio/ide/how-to-view-and-edit-code-by-using-peek-definition-alt-plus-f12?view=vs-2019)

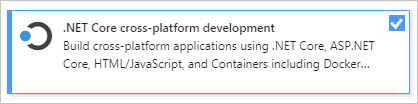
The **Peek Definition** window shows the definition of a method or type without actually opening a separate file.



**Install the Visual Studio IDE**

In this section, you'll create a simple project to try out some of the things you can do with Visual Studio. You'll use [IntelliSense](https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2019) as a coding aid, debug an app to see the value of a variable during the program's execution, and change the color theme.

To get started, [download Visual Studio](https://visualstudio.microsoft.com/downloads) and install it on your system. The modular installer enables you to choose and install *workloads*, which are groups of features needed for the programming language or platform you prefer. To follow the steps for [creating a program](https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019#create-a-program), be sure to select the **.NET Core cross-platform development** workload during installation.



When you open Visual Studio for the first time, you can optionally [sign in](https://docs.microsoft.com/en-us/visualstudio/ide/signing-in-to-visual-studio?view=vs-2019) using your Microsoft account or your work or school account.

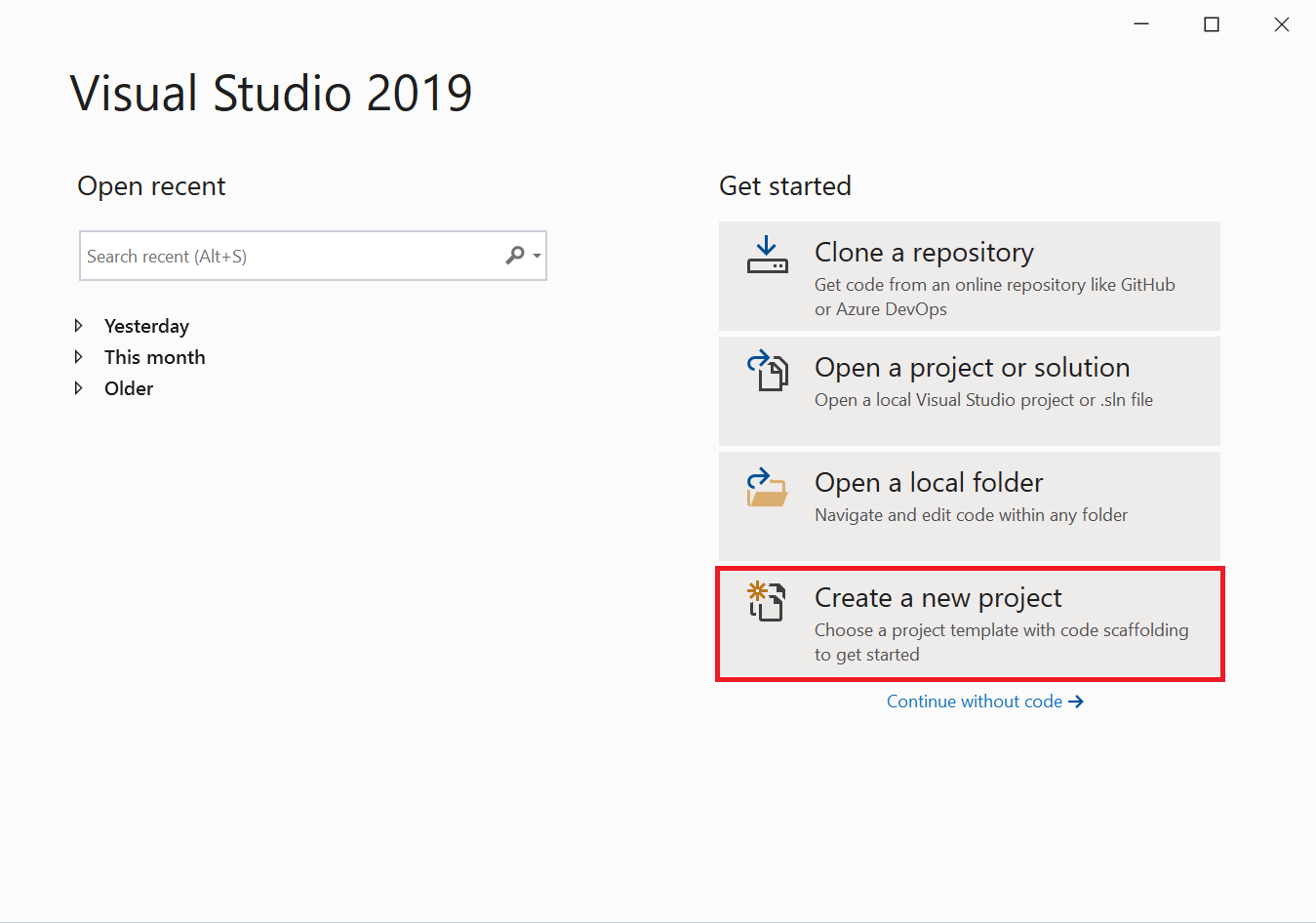
**Create a program**

Let's dive in and create a simple program.

1. Open Visual Studio.

The start window appears with various options for cloning a repo, opening a recent project, or creating a brand new project.

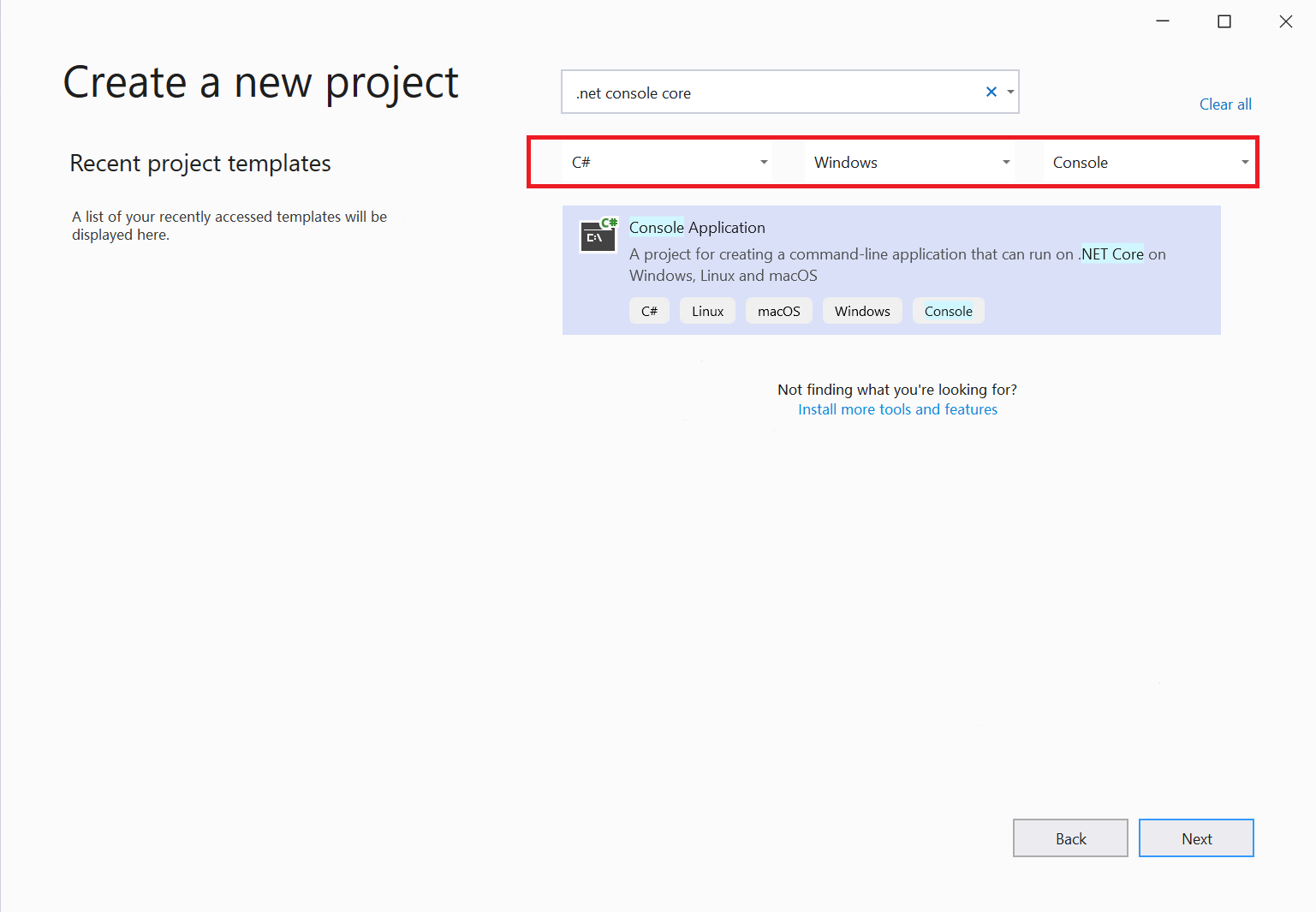
1. Choose **Create a new project**.



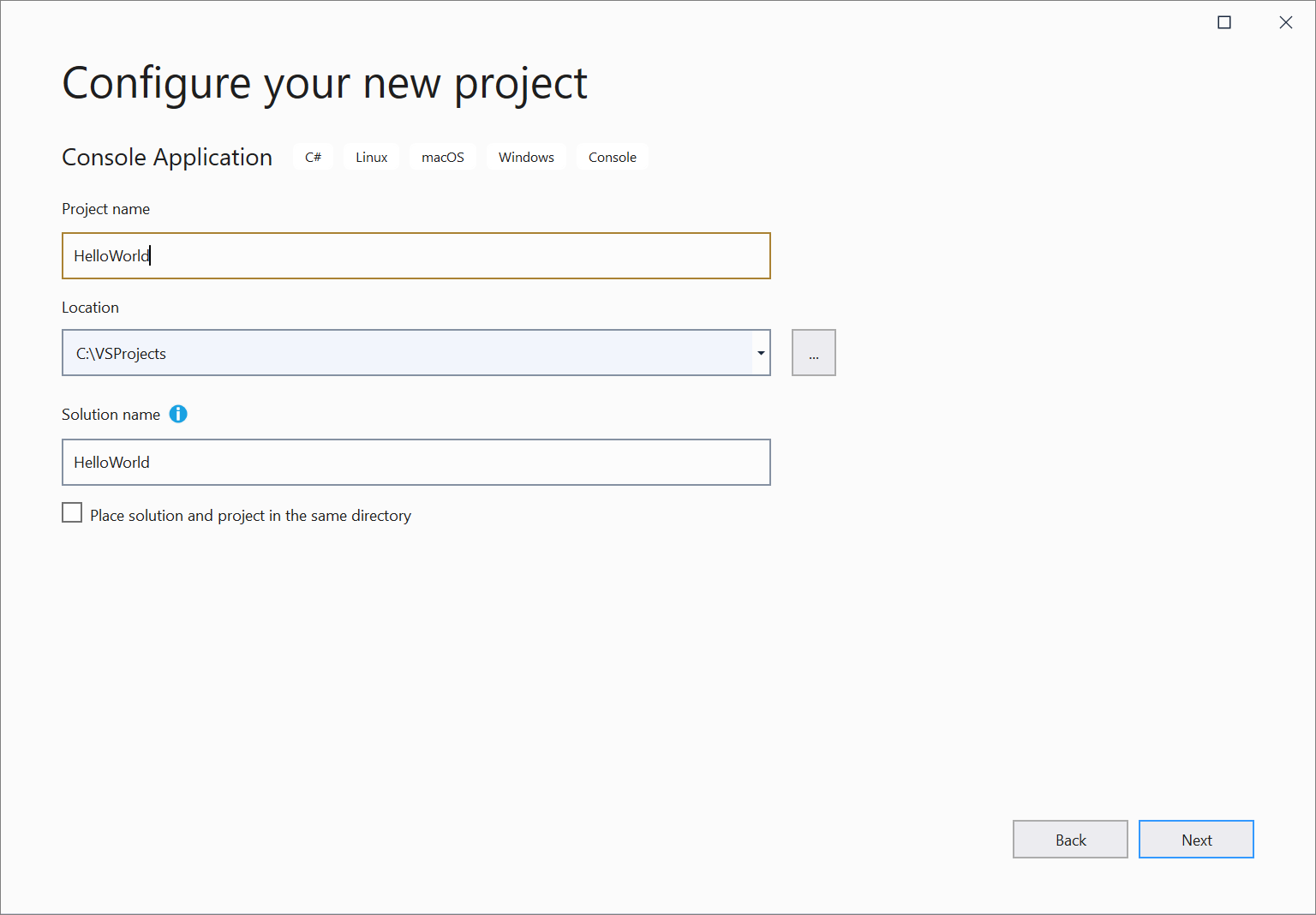
The **Create a new project** window opens and shows several project *templates*. A template contains the basic files and settings required for a given project type.

1. To find the template we want, type or enter **.net core console** in the search box. The list of available templates is automatically filtered based on the keywords you entered. You can further filter the template results by choosing **C#** from the **All language** drop-down list, **Windows** from the **All platforms** list, and **Console** from the **All project types** list .

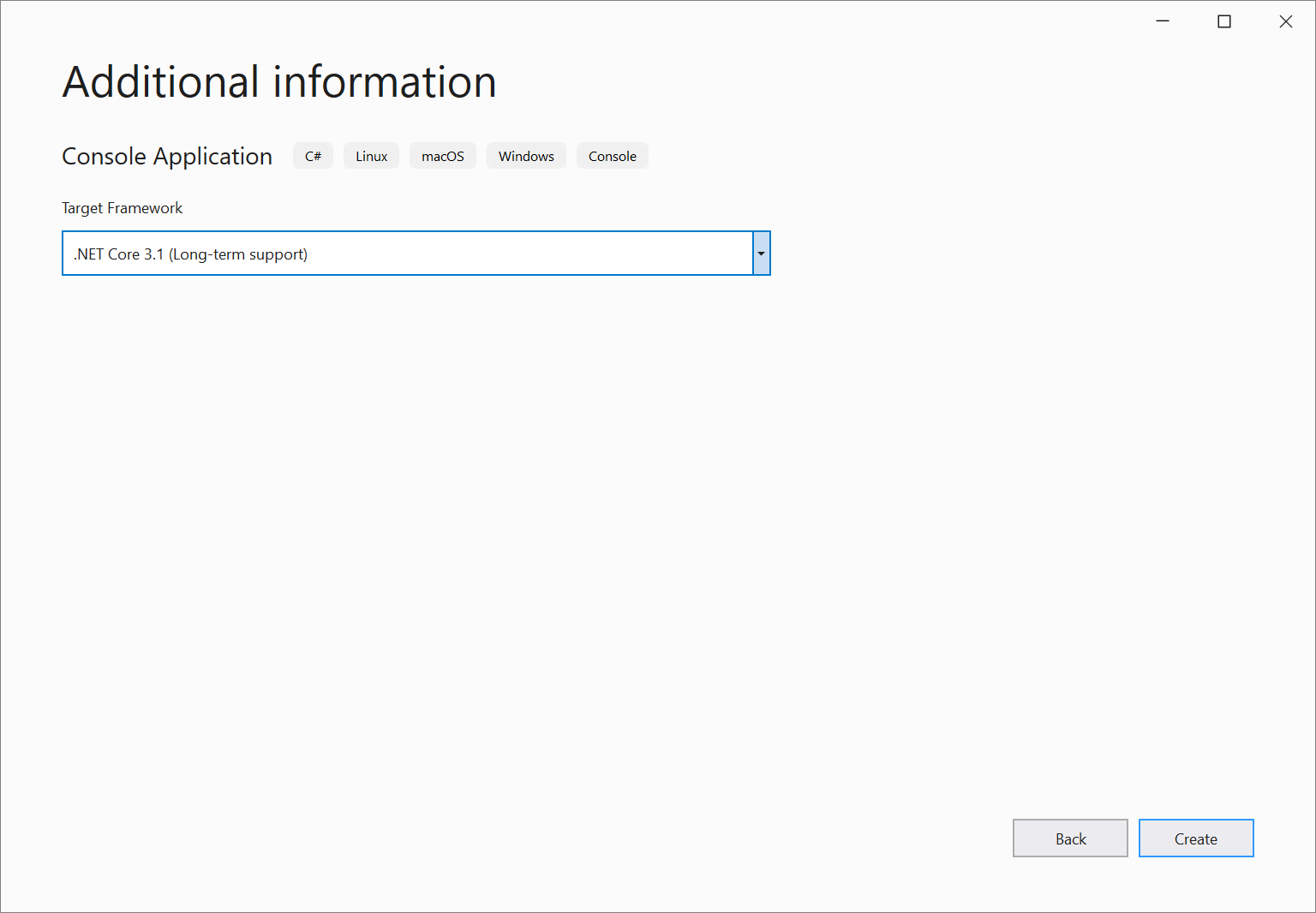
Select the **Console Application** template, and then click **Next**.



1. In the **Configure your new project** window, enter **HelloWorld** in the **Project name** box, optionally change the directory location for your project files (the default locale is C:\Users\<name>\source\repos), and then click **Next**.

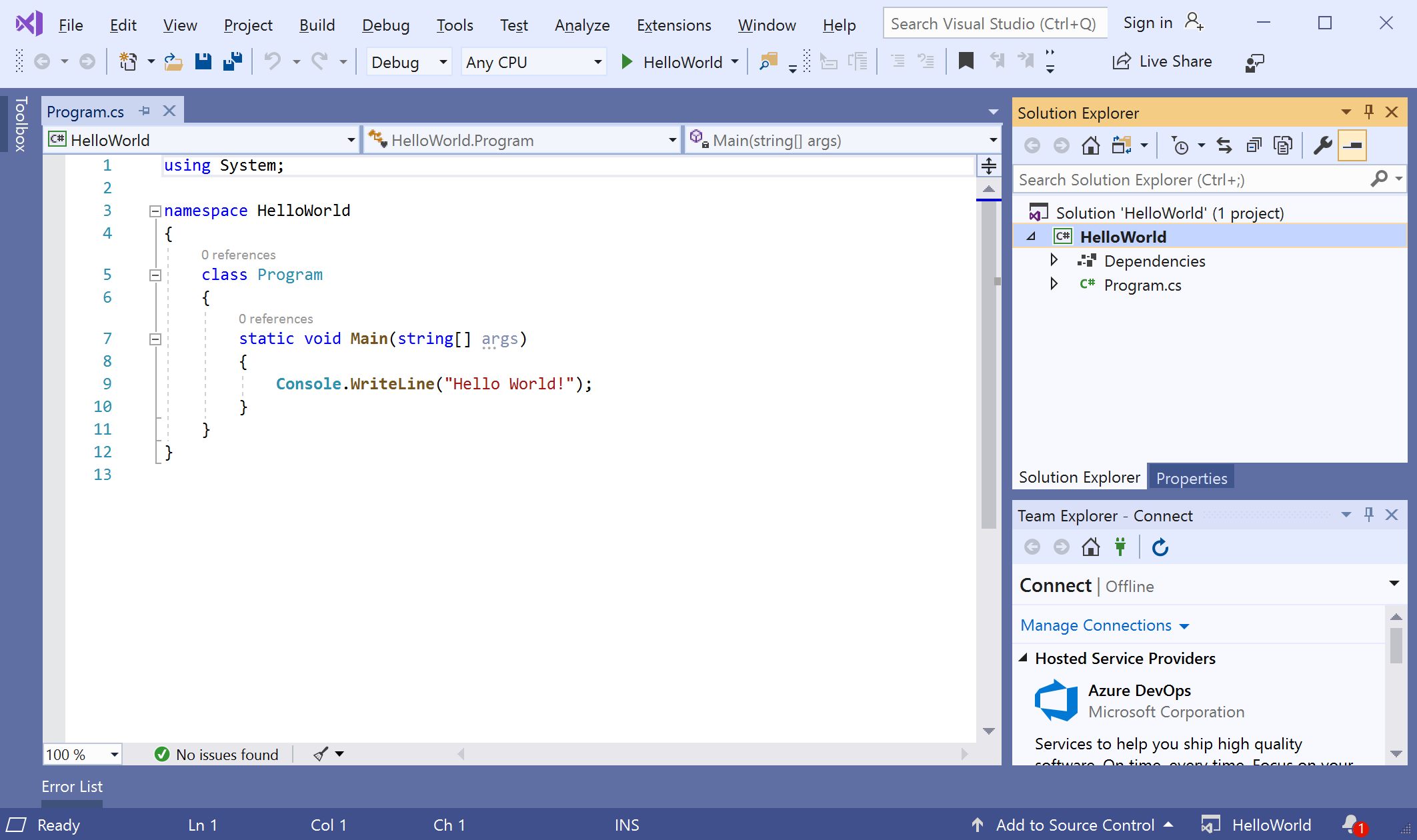


1. In the **Additional information** window, verify that **.NET Core 3.1** appears in the **Target Framework** drop-down menu, and then click **Create**.

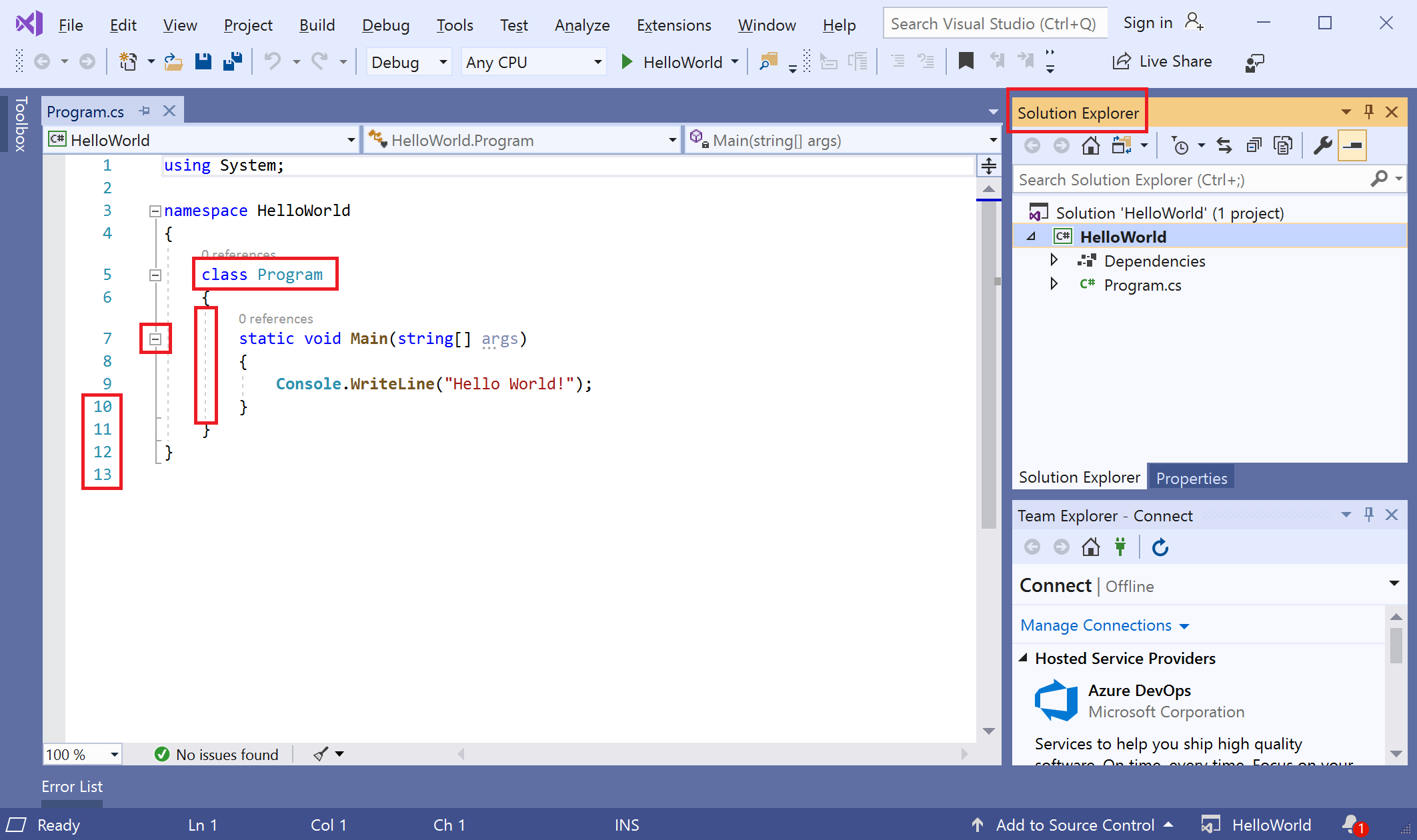


Visual Studio creates the project. It's a simple "Hello World" application that calls the [Console.WriteLine()](https://docs.microsoft.com/en-us/dotnet/api/system.console.writeline" \l "System_Console_WriteLine) method to display the literal string "Hello World!" in the console (program output) window.

Shortly, you should see something like the following:

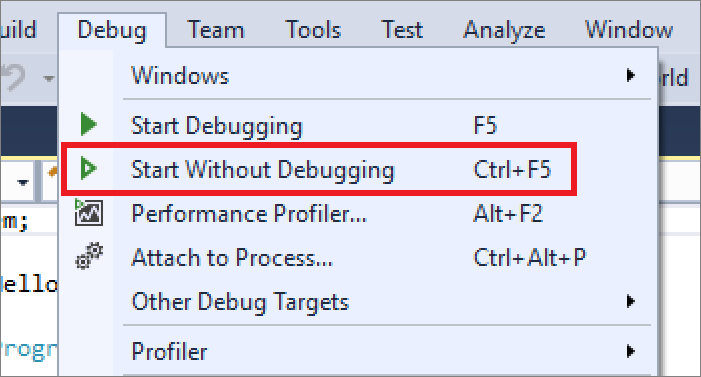


The C# code for your application shows in the editor window, which takes up most of the space. Notice that the text is automatically colorized to indicate different parts of the code, such as keywords and types. In addition, small, vertical dashed lines in the code indicate which braces match one another, and line numbers help you locate code later. You can choose the small, boxed minus signs to collapse or expand blocks of code. This code outlining feature lets you hide code you don't need, helping to minimize onscreen clutter. The project files are listed on the right side in a window called **Solution Explorer**.

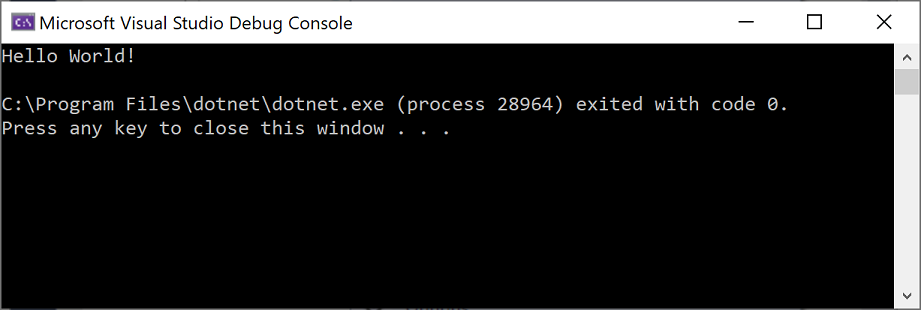


There are other menus and tool windows available, but let's move on for now.

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



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



1. To close the console window, press any key on your keyboard.
2. Let's add some additional code to the app. Add the following C# code before the line that says Console.WriteLine("Hello World!");:

C#Copy

Console.WriteLine("\nWhat is your name?");

var name = Console.ReadLine();

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

1. Change the line that says Console.WriteLine("Hello World!"); to the following code:

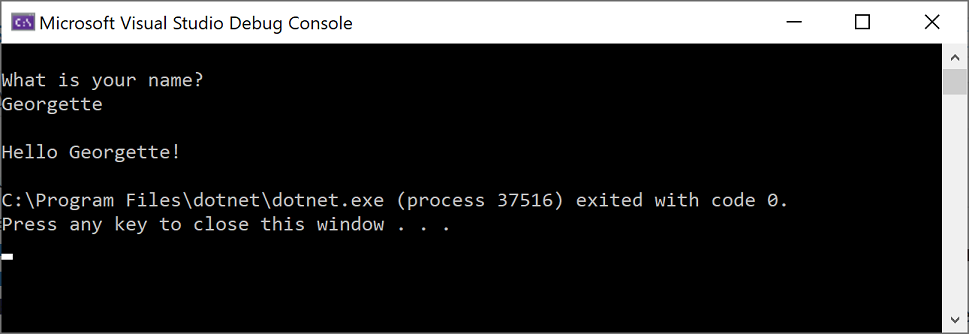
C#Copy

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

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

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

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



1. Press any key to close the console window and stop the running program.

**Use refactoring and IntelliSense**

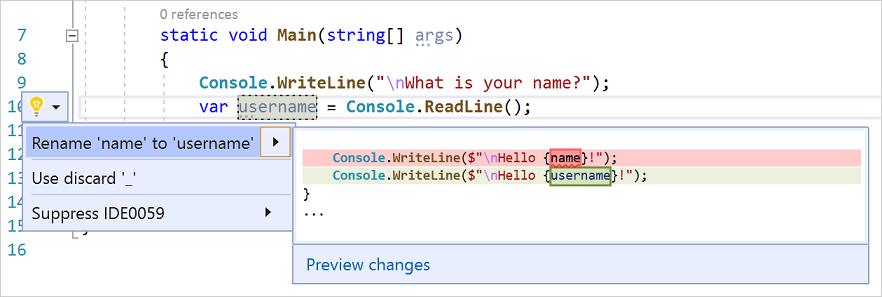
Let's look at a couple of the ways that [refactoring](https://docs.microsoft.com/en-us/visualstudio/ide/refactoring-in-visual-studio?view=vs-2019) and [IntelliSense](https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2019) can help you code more efficiently.

First, let's rename the name variable:

1. Double-click the name variable to select it.
2. Type in the new name for the variable, **username**.

Notice that a gray box appears around the variable, and a light bulb appears in the margin.

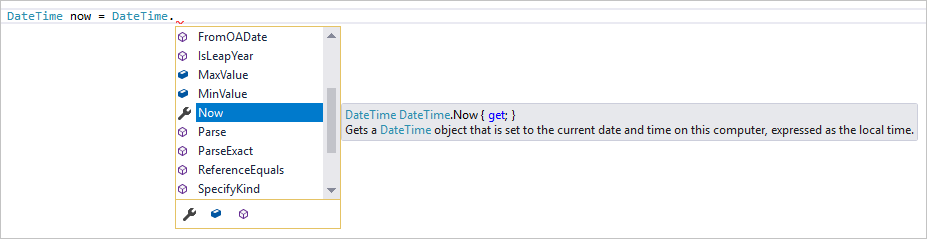
1. Select the light bulb icon to show the available [Quick Actions](https://docs.microsoft.com/en-us/visualstudio/ide/quick-actions?view=vs-2019). Select **Rename 'name' to 'username'**.



The variable is renamed across the project, which in our case is only two places.

1. Now let's take a look at IntelliSense. Below the line that says Console.WriteLine($"\nHello {username}!");, type DateTime now = DateTime..

A box displays the members of the [DateTime](https://docs.microsoft.com/en-us/dotnet/api/system.datetime) class. In addition, the description of the currently selected member displays in a separate box.



1. Select the member named **Now**, which is a property of the class, by double-clicking on it or pressing **Tab**. Complete the line of code by adding a semi-colon to the end.
2. Below that, type in or paste the following lines of code:

C#Copy

int dayOfYear = now.DayOfYear;

Console.Write("Day of year: ");

Console.WriteLine(dayOfYear);

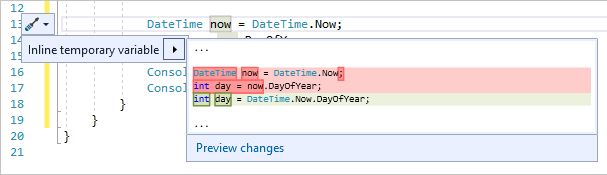
**Tip**

[**Console.Write**](https://docs.microsoft.com/en-us/dotnet/api/system.console.write) is a little different to **[Console.WriteLine](https://docs.microsoft.com/en-us/dotnet/api/system.console.writeline)** in that it doesn't add a line terminator after it prints. That means that the next piece of text that's sent to the output will print on the same line. You can hover over each of these methods in your code to see their description.

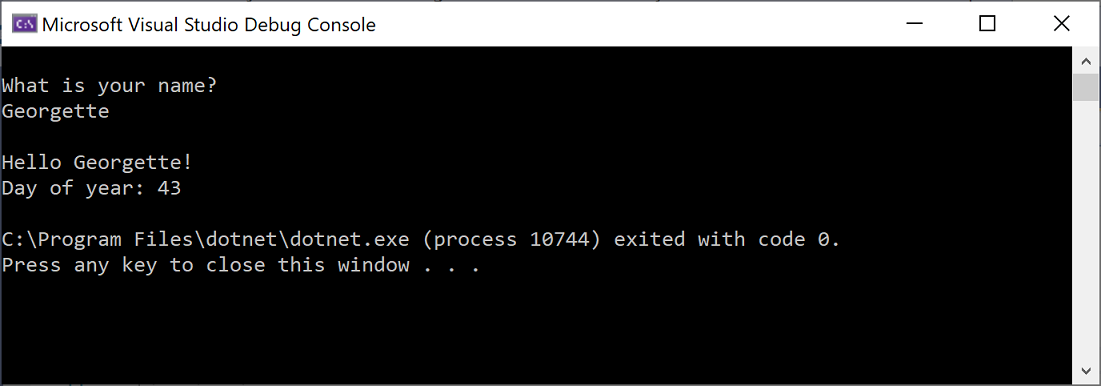
1. Next, we'll use refactoring again to make the code a little more concise. Click on the variable now in the line DateTime now = DateTime.Now;.

Notice that a little screwdriver icon appears in the margin on that line.

1. Click the screwdriver icon to see what suggestions Visual Studio has available. In this case, it's showing the [Inline temporary variable](https://docs.microsoft.com/en-us/visualstudio/ide/reference/inline-temporary-variable?view=vs-2019) refactoring to remove a line of code without changing the overall behavior of the code:



1. Click **Inline temporary variable** to refactor the code.
2. Run the program again by pressing **Ctrl**+**F5**. The output looks something like this:



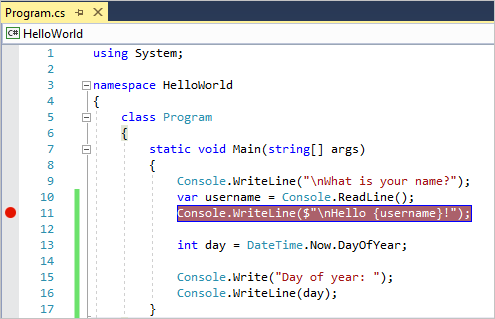
**Debug code**

When you write code, you need to run it and test it for bugs. Visual Studio's debugging system lets you step through code one statement at a time and inspect variables as you go. You can set *breakpoints* that stop execution of the code at a particular line. You can observe how the value of a variable changes as the code runs, and more.

Let's set a breakpoint to see the value of the username variable while the program is "in flight".

1. Find the line of code that says Console.WriteLine($"\nHello {username}!");. To set a breakpoint on this line of code, that is, to make the program pause execution at this line, click in the far left margin of the editor. You can also click anywhere on the line of code and then press **F9**.

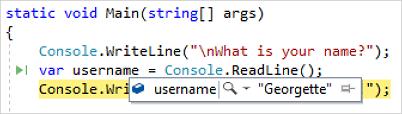
A red circle appears in the far left margin, and the code is highlighted in red.



1. Start debugging by selecting **Debug** > **Start Debugging** or by pressing **F5**.
2. When the console window appears and asks for your name, type it in and press **Enter**.

The focus returns to the Visual Studio code editor and the line of code with the breakpoint is highlighted in yellow. This signifies that it's the next line of code that the program will execute.

1. Hover your mouse over the username variable to see its value. Alternatively, you can right-click on username and select **Add Watch** to add the variable to the **Watch** window, where you can also see its value.



1. To let the program run to completion, press **F5** again.

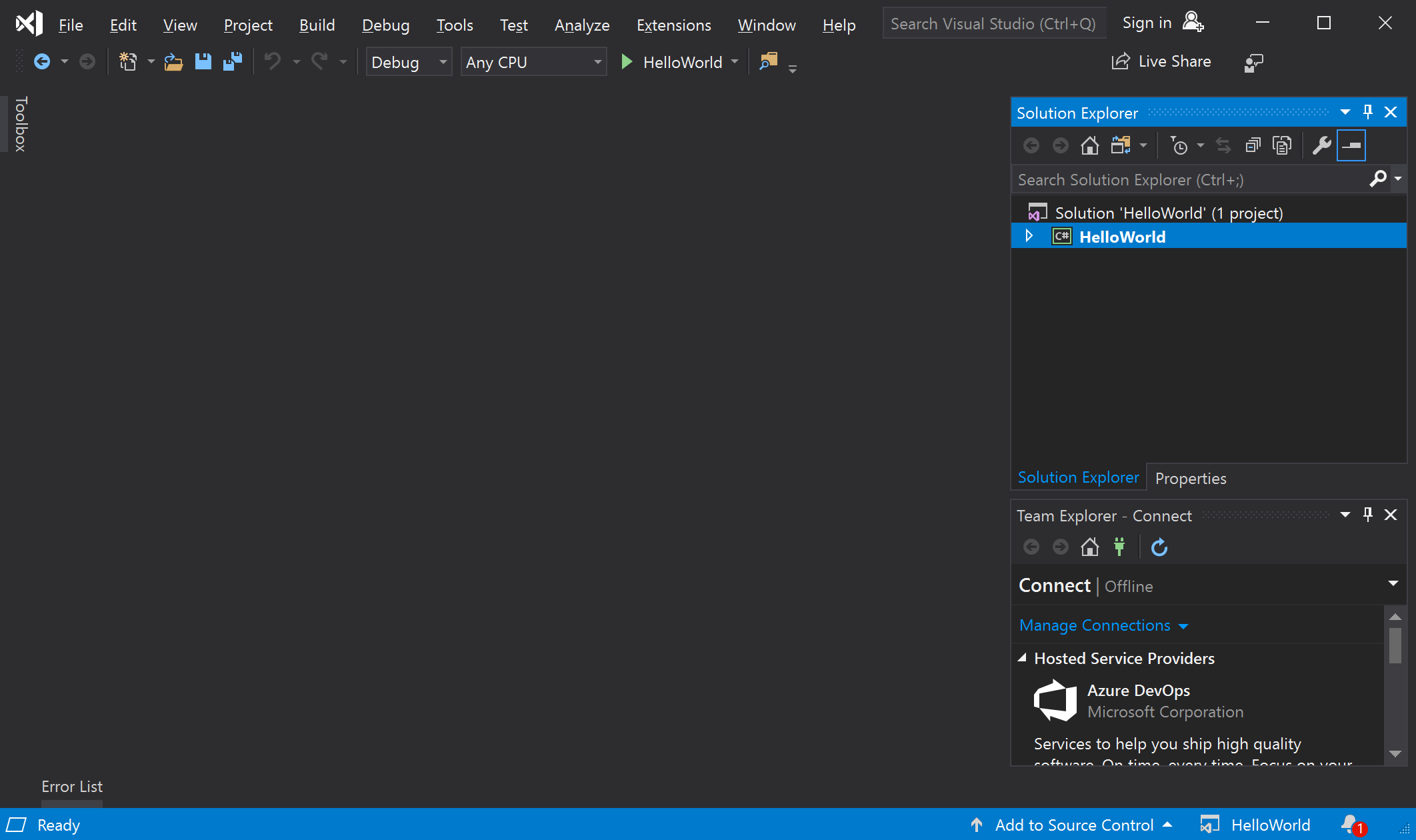
To get more details about debugging in Visual Studio, see [Debugger feature tour](https://docs.microsoft.com/en-us/visualstudio/debugger/debugger-feature-tour?view=vs-2019).

**Customize Visual Studio**

You can personalize the Visual Studio user interface, including change the default color theme. To change to the **Dark** theme:

1. On the menu bar, choose **Tools** > **Options** to open the **Options** dialog.
2. On the **Environment** > **General** options page, change the **Color theme** selection to **Dark**, and then choose **OK**.

The color theme for the entire IDE changes to **Dark**.



To learn about other ways you can personalize the IDE, see [Personalize Visual Studio](https://docs.microsoft.com/en-us/visualstudio/ide/personalizing-the-visual-studio-ide?view=vs-2019).

**Next steps**

Explore Visual Studio further by following along with one of these introductory articles:

* Get acquainted with the code editor in [Learn to use the code editor](https://docs.microsoft.com/en-us/visualstudio/get-started/tutorial-editor?view=vs-2019)
* Learn how Visual Studio organizes code in [Learn about projects and solutions](https://docs.microsoft.com/en-us/visualstudio/get-started/tutorial-projects-solutions?view=vs-2019)

If you're ready to dive into more coding, one of the following language-specific quickstarts is a good next step:

* [Use Visual Studio to create your first Python web app](https://docs.microsoft.com/en-us/visualstudio/ide/quickstart-python?view=vs-2019)
* [Use Visual Studio to create your first C# web app](https://docs.microsoft.com/en-us/visualstudio/ide/quickstart-aspnet-core?view=vs-2019)
* [Use Visual Studio to create your first F# web app](https://docs.microsoft.com/en-us/visualstudio/ide/quickstart-fsharp?view=vs-2019)
* [Use Visual Studio to create your first Node.js app](https://docs.microsoft.com/en-us/visualstudio/ide/quickstart-nodejs?view=vs-2019)
* [Use Visual Studio to create your first C++ console app](https://docs.microsoft.com/en-us/cpp/get-started/tutorial-console-cpp)

**See also**

* Discover [more Visual Studio features](https://docs.microsoft.com/en-us/visualstudio/ide/advanced-feature-overview?view=vs-2019)
* Visit [visualstudio.microsoft.com](https://visualstudio.microsoft.com/vs/)
* Read [The Visual Studio blog](https://devblogs.microsoft.com/visualstudio/)

**Recommended content**

**[Introduction to editing in the code editor - Visual Studio](https://docs.microsoft.com/en-us/visualstudio/get-started/tutorial-editor)**

Learn how to use the code editor in Visual Studio to add code to a file, and also how to write code, navigate to it, and refactor it.

**[Quickstart: Tour of the Visual Studio IDE](https://docs.microsoft.com/en-us/visualstudio/ide/quickstart-ide-orientation)**

Learn about some of the windows, menus, and other UI features of the Visual Studio integrated development environment (IDE).

**[Create a console calculator in C++](https://docs.microsoft.com/en-us/cpp/get-started/tutorial-console-cpp)**

Create a Hello World console app and a calculator app in Visual C++

**[Python in Visual Studio tutorial step 0, installation](https://docs.microsoft.com/en-us/visualstudio/python/tutorial-working-with-python-in-visual-studio-step-00-installation)**

Step 0 (installation prerequisites) of a core walkthrough of working with Python in Visual Studio.

Show more

**interface (C# Reference)**

* 01/17/2020
* 3 minutes to read
  + [](https://github.com/BillWagner)
  + [](https://github.com/MightyPen)
  + [](https://github.com/pkulikov)
  + [](https://github.com/gewarren)
  + [](https://github.com/mairaw)
  + +6

An interface defines a contract. Any [class](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) or [struct](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct) that implements that contract must provide an implementation of the members defined in the interface. Beginning with C# 8.0, an interface may define a default implementation for members. It may also define [static](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static) members in order to provide a single implementation for common functionality.

In the following example, class ImplementationClass must implement a method named SampleMethod that has no parameters and returns void.

For more information and examples, see [Interfaces](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/interfaces).

**Example interface**

C#Copy

interface ISampleInterface

{

void SampleMethod();

}

class ImplementationClass : ISampleInterface

{

// Explicit interface member implementation:

void ISampleInterface.SampleMethod()

{

// Method implementation.

}

static void Main()

{

// Declare an interface instance.

ISampleInterface obj = new ImplementationClass();

// Call the member.

obj.SampleMethod();

}

}

An interface can be a member of a namespace or a class. An interface declaration can contain declarations (signatures without any implementation) of the following members:

* [Methods](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods)
* [Properties](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties)
* [Indexers](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/indexers/using-indexers)
* [Events](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/event)

These preceding member declarations typically do not contain a body. Beginning with C# 8.0, an interface member may declare a body. This is called a *default implementation*. Members with bodies permit the interface to provide a "default" implementation for classes and structs that don't provide an overriding implementation. In addition, beginning with C# 8.0, an interface may include:

* [Constants](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/const)
* [Operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/operator-overloading)
* [Static constructor](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constructors#static-constructors).
* [Nested types](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/nested-types)
* [Static fields, methods, properties, indexers, and events](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static)
* Member declarations using the explicit interface implementation syntax.
* Explicit access modifiers (the default access is [public](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/access-modifiers)).

Interfaces may not contain instance state. While static fields are now permitted, instance fields are not permitted in interfaces. [Instance auto-properties](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties) are not supported in interfaces, as they would implicitly declare a hidden field. This rule has a subtle effect on property declarations. In an interface declaration, the following code does not declare an auto-implemented property as it does in a class or struct. Instead, it declares a property that doesn't have a default implementation but must be implemented in any type that implements the interface:

C#Copy

public interface INamed

{

public string Name {get; set;}

}

An interface can inherit from one or more base interfaces. When an interface [overrides a method](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override) implemented in a base interface, it must use the [explicit interface implementation](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/explicit-interface-implementation) syntax.

When a base type list contains a base class and interfaces, the base class must come first in the list.

A class that implements an interface can explicitly implement members of that interface. An explicitly implemented member cannot be accessed through a class instance, but only through an instance of the interface. In addition, default interface members can only be accessed through an instance of the interface.

For more information about explicit interface implementation, see [Explicit Interface Implementation](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/explicit-interface-implementation).

**Example interface implementation**

The following example demonstrates interface implementation. In this example, the interface contains the property declaration and the class contains the implementation. Any instance of a class that implements IPoint has integer properties x and y.

C#Copy

interface IPoint

{

// Property signatures:

int X

{

get;

set;

}

int Y

{

get;

set;

}

double Distance

{

get;

}

}

class Point : IPoint

{

// Constructor:

public Point(int x, int y)

{

X = x;

Y = y;

}

// Property implementation:

public int X { get; set; }

public int Y { get; set; }

// Property implementation

public double Distance =>

Math.Sqrt(X \* X + Y \* Y);

}

class MainClass

{

static void PrintPoint(IPoint p)

{

Console.WriteLine("x={0}, y={1}", p.X, p.Y);

}

static void Main()

{

IPoint p = new Point(2, 3);

Console.Write("My Point: ");

PrintPoint(p);

}

}

// Output: My Point: x=2, y=3

**C# language specification**

For more information, see the [Interfaces](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/interfaces) section of the [C# language specification](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction) and the feature specification for [Default interface members - C# 8.0](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-8.0/default-interface-methods)

**See also**

* [C# Reference](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/)
* [C# Programming Guide](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/)
* [C# Keywords](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/)
* [Reference Types](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types)
* [Interfaces](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/interfaces)
* [Using Properties](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties)
* [Using Indexers](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/indexers/using-indexers)