

Get started with unit testing

03/31/2019 • 3 minutes to read •  +2

In this article

[Create unit tests](#)

[Run unit tests](#)

[View live unit test results](#)

[Generate unit tests with IntelliTest](#)

[Analyze code coverage](#)

[Use a third-party test framework](#)

[See also](#)


Use Visual Studio to define and run unit tests to maintain code health, ensure code coverage, and find errors and faults before your customers do. Run your unit tests frequently to make sure your code is working properly.

Create unit tests

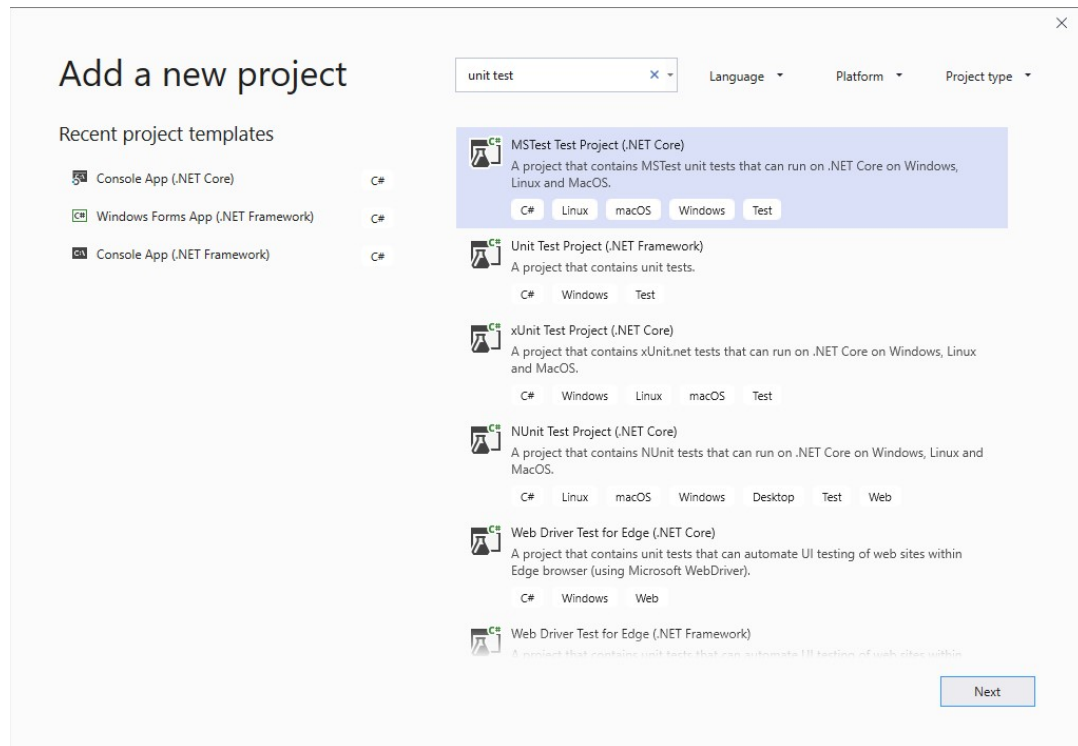
This section describes at a high level how to create a unit test project.

1. Open the project that you want to test in Visual Studio.

For the purposes of demonstrating an example unit test, this article tests a simple "Hello World" project. The sample code for such a project is as follows:

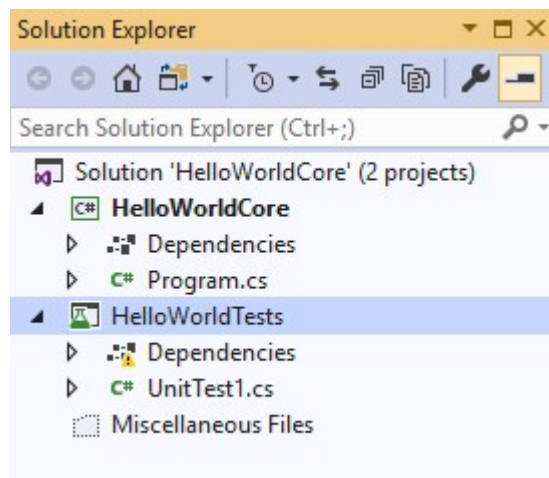
C#	 Copy
<pre>public class Program { public static void Main() { Console.WriteLine("Hello World!"); } }</pre>	

2. In **Solution Explorer**, select the solution node. Then, from the top menu bar, select **File > Add > New Project**.
3. In the new project dialog box, find a unit test project template for the test framework you want to use and select it.

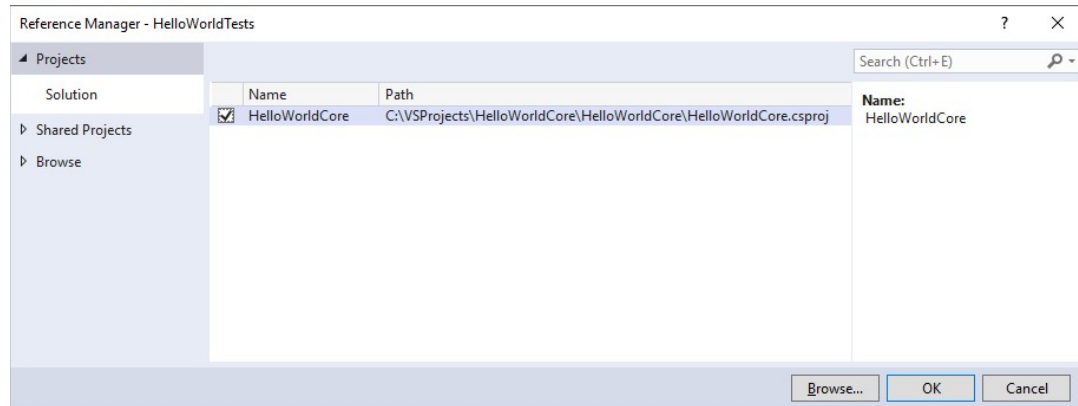


Click **Next**, choose a name for the test project, and then click **Create**.

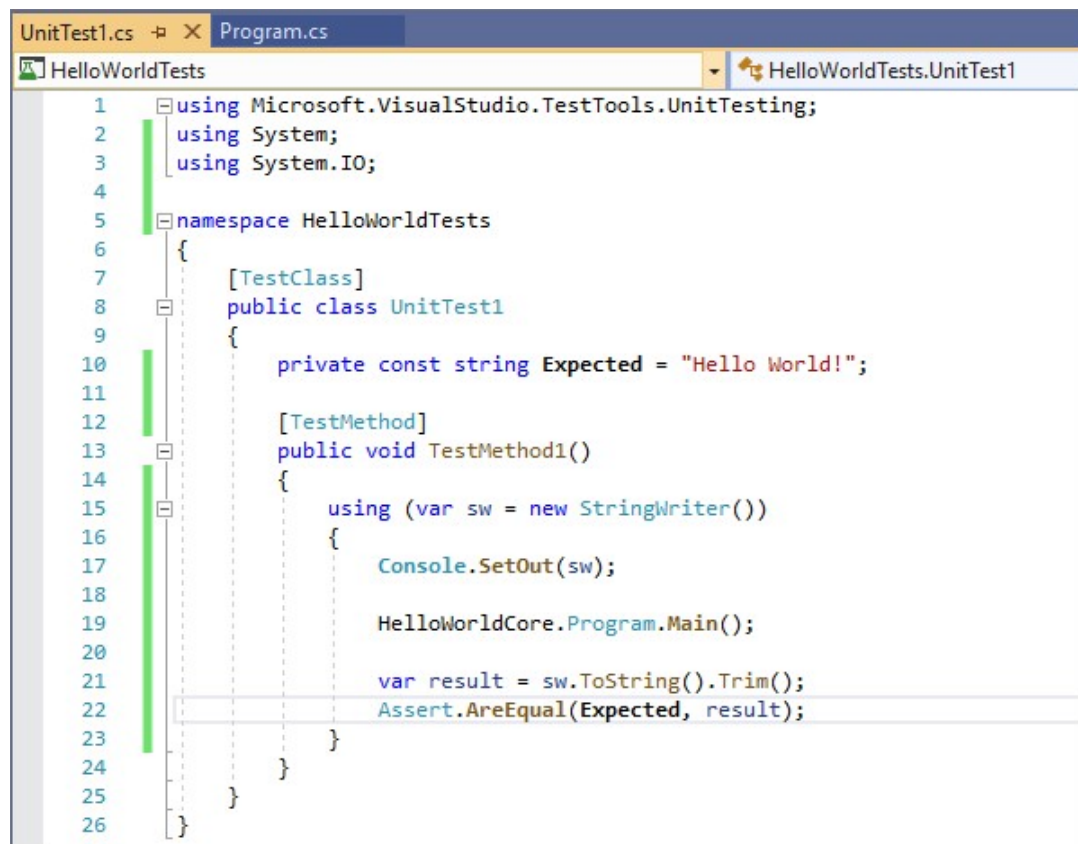
The project is added to your solution.



4. In the unit test project, add a reference to the project you want to test by right-clicking on **References** or **Dependencies** and then choosing **Add Reference**.
5. Select the project that contains the code you'll test and click **OK**.



6. Add code to the unit test method.

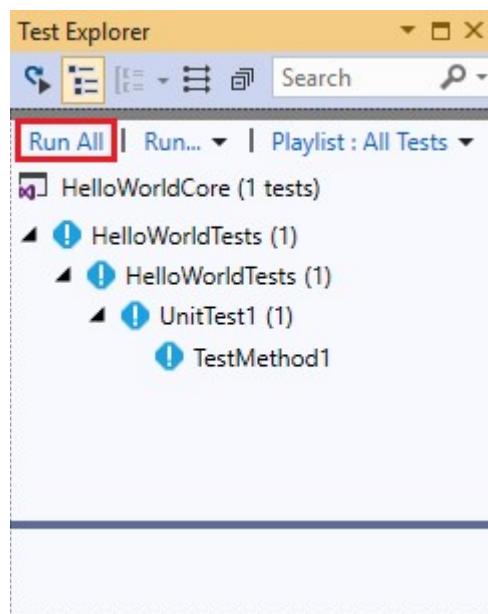


 **Tip**

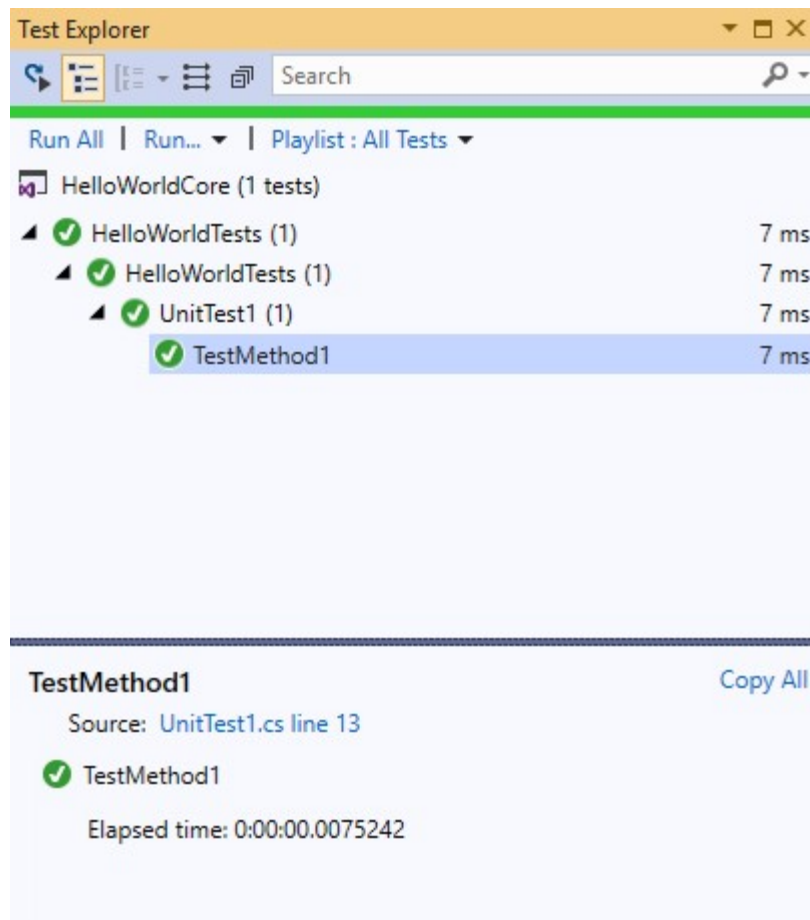
For a more detailed walkthrough of creating unit tests, see [Create and run unit tests for managed code](#).

Run unit tests

1. Open [Test Explorer](#) by choosing **Test > Windows > Test Explorer** from the top menu bar.
2. Run your unit tests by clicking **Run All**.



After the tests have completed, a green check mark indicates that a test passed. A red "x" icon indicates that a test failed.



Tip

You can use [Test Explorer](#) to run unit tests from the built-in test framework (MSTest) or from third-party test frameworks. You can group tests into categories, filter the test list, and create, save, and run playlists of tests. You can also debug tests and analyze test performance and code coverage.

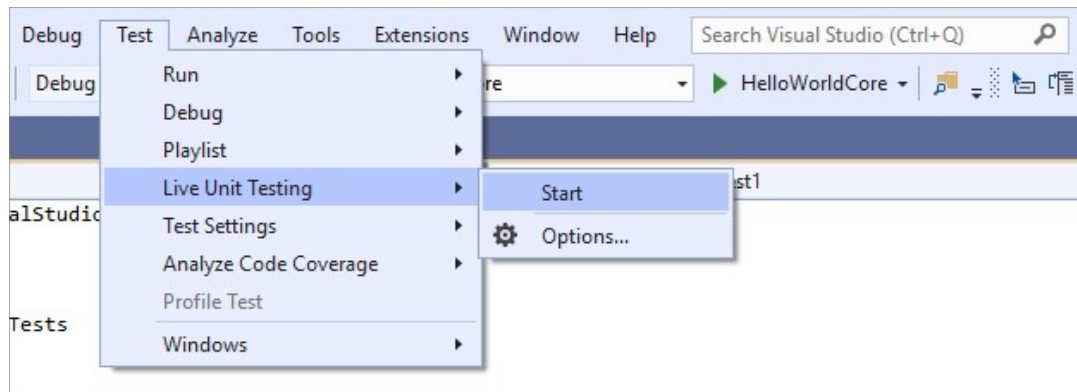
View live unit test results

If you are using the MSTest, xUnit, or NUnit testing framework in Visual Studio 2017 or later, you can see live results of your unit tests.

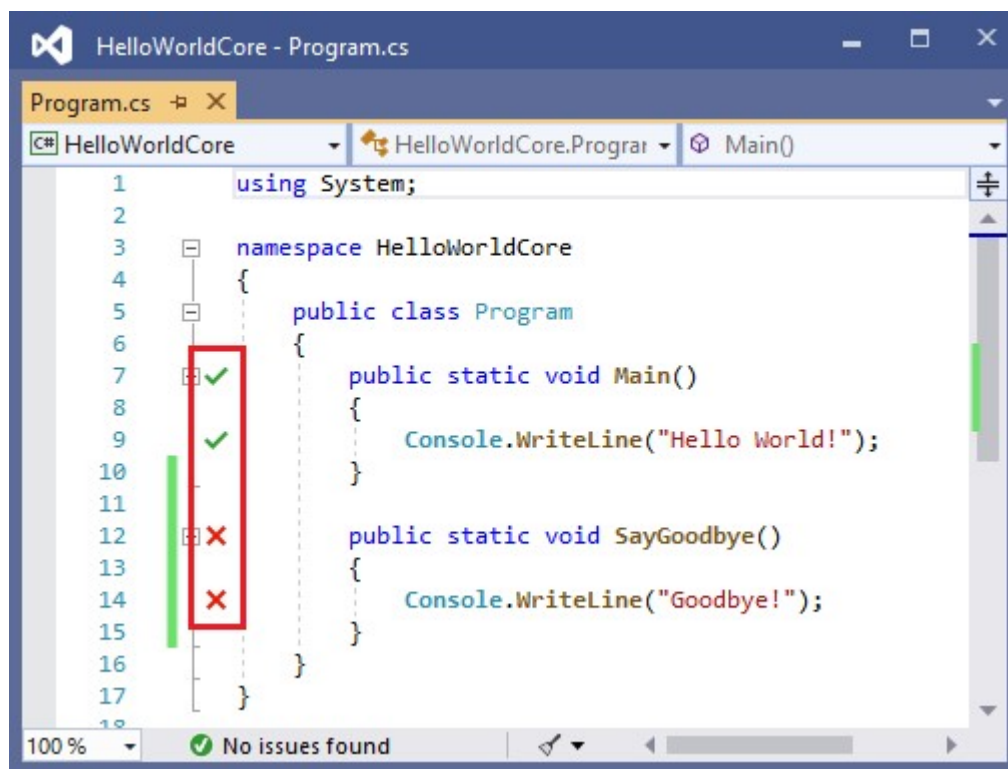
Note

Live unit testing is available in Enterprise edition only.

1. Turn live unit testing from the **Test** menu by choosing **Test > Live Unit Testing > Start**.



2. View the results of the tests within the code editor window as you write and edit code.



3. Click a test result indicator to see more information, such as the names of the tests that cover that method.



For more information about live unit testing, see [Live unit testing](#).

Generate unit tests with IntelliTest

When you run IntelliTest, you can see which tests are failing and add any necessary code to fix them. You can select which of the generated tests to save into a test project to provide a regression suite. As you change your code, rerun IntelliTest to keep the generated tests in sync with your code changes. To learn how, see [Generate unit tests for your code with IntelliTest](#).

💡 Tip

IntelliTest is only available for managed code that targets the .NET Framework.

IntelliTest Exploration Results - stopped				
Triangle.ClassifyBySideLengths(int)				
Run 0 Warnings				
<div> ✓ 8 ✗ 4 16/16 blocks, 0/0 asserts, 12 runs </div>				
	lengths	result	Summary/Exception	Error Message
✗ 1	null		NullReferenceException	Object refer...
✗ 2	{}		IndexOutOfRangeException	Index was out...
✗ 3	{0}		IndexOutOfRangeException	Index was out...
✗ 4	{0, 0}		IndexOutOfRangeException	Index was out...
✓ 5	{0, 0, 0}	Invalid		
✓ 6	{5, 538, 0}	Invalid		
✓ 7	{67, 0, 0}	Invalid		
✓ 8	{422, 536, 6...}	Scalene		
✓ 9	{528, 413, 5...}	Isosceles		
✓ 10	{2, 2, 3}	Isosceles		
✓ 11	{1, 512, 512}	Isosceles		
✓ 12	{512, 512, 5...}	Equilateral		

Details:
Stack trace:
 System.NullReferenceException...
 at Triangle.ClassifyBySideLengt...
 at TriangleTest.ClassifyBySideLe...

Analyze code coverage

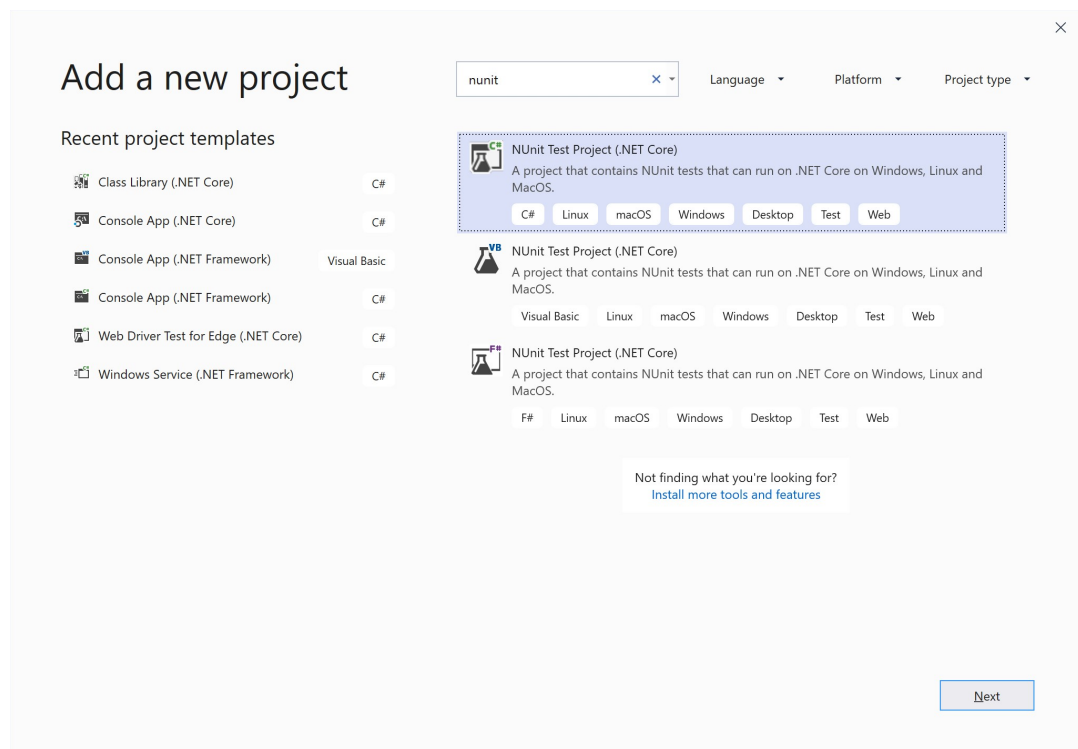
To determine what proportion of your project's code is actually being tested by coded tests such as unit tests, you can use the code coverage feature of Visual Studio. To guard effectively against bugs, your tests should exercise a large proportion of your code. To learn how, see [Use code coverage to determine how much code is being tested](#).

Use a third-party test framework

You can run unit tests in Visual Studio by using third-party test frameworks such as Boost, Google, and NUnit. Use the **NuGet Package Manager** to install the NuGet package for the framework of your choice. Or, for the NUnit and xUnit test frameworks, Visual Studio includes preconfigured test project templates that include the necessary NuGet packages.

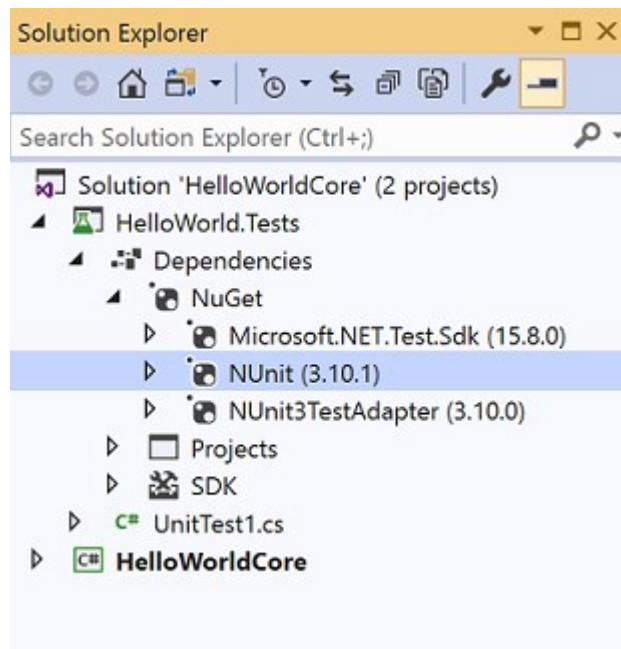
To create unit tests that use [NUnit](#):

1. Open the solution that contains the code you want to test.
2. Right-click on the solution in **Solution Explorer** and choose **Add > New Project**.
3. Select the **NUnit Test Project** project template.



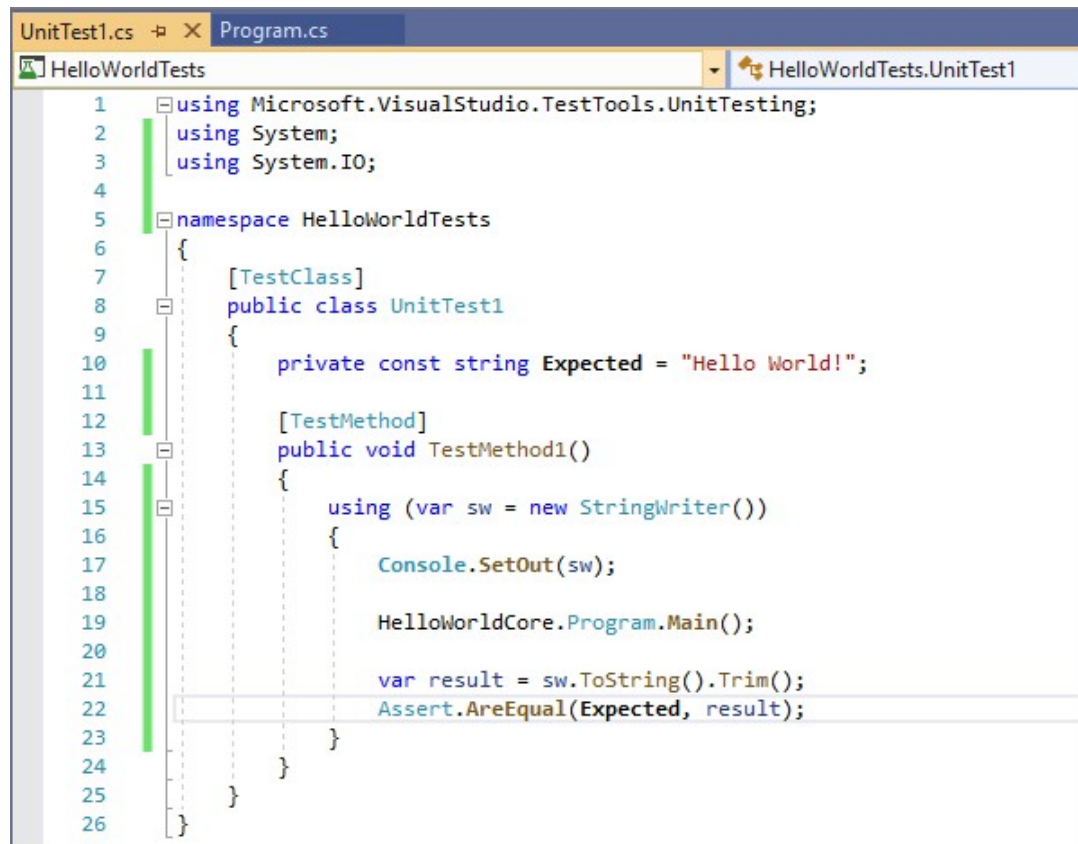
Click **Next**, name the project, and then click **Create**.

The project template includes NuGet references to NUnit and NUnit3TestAdapter.



4. Add a reference from the test project to the project that contains the code you want to test.

5. Add code to your test method.



```
1  using Microsoft.VisualStudio.TestTools.UnitTesting;
2  using System;
3  using System.IO;
4
5  namespace HelloWorldTests
6  {
7      [TestClass]
8      public class UnitTest1
9      {
10         private const string Expected = "Hello World!";
11
12         [TestMethod]
13         public void TestMethod1()
14         {
15             using (var sw = new StringWriter())
16             {
17                 Console.SetOut(sw);
18
19                 HelloWorldCore.Program.Main();
20
21                 var result = sw.ToString().Trim();
22                 Assert.AreEqual(Expected, result);
23             }
24         }
25     }
26 }
```

6. Run the test from **Test Explorer** or by right-clicking on the test code and choosing **Run Test(s)**.

See also

- [Walkthrough: Create and run unit tests for managed code](#)
- [Create Unit Tests command](#)
- [Generate tests with IntelliTest](#)
- [Run tests with Test Explorer](#)
- [Analyze code coverage](#)