

Supplement 1

Using Visual Studio 2026

Using Visual Studio 2026

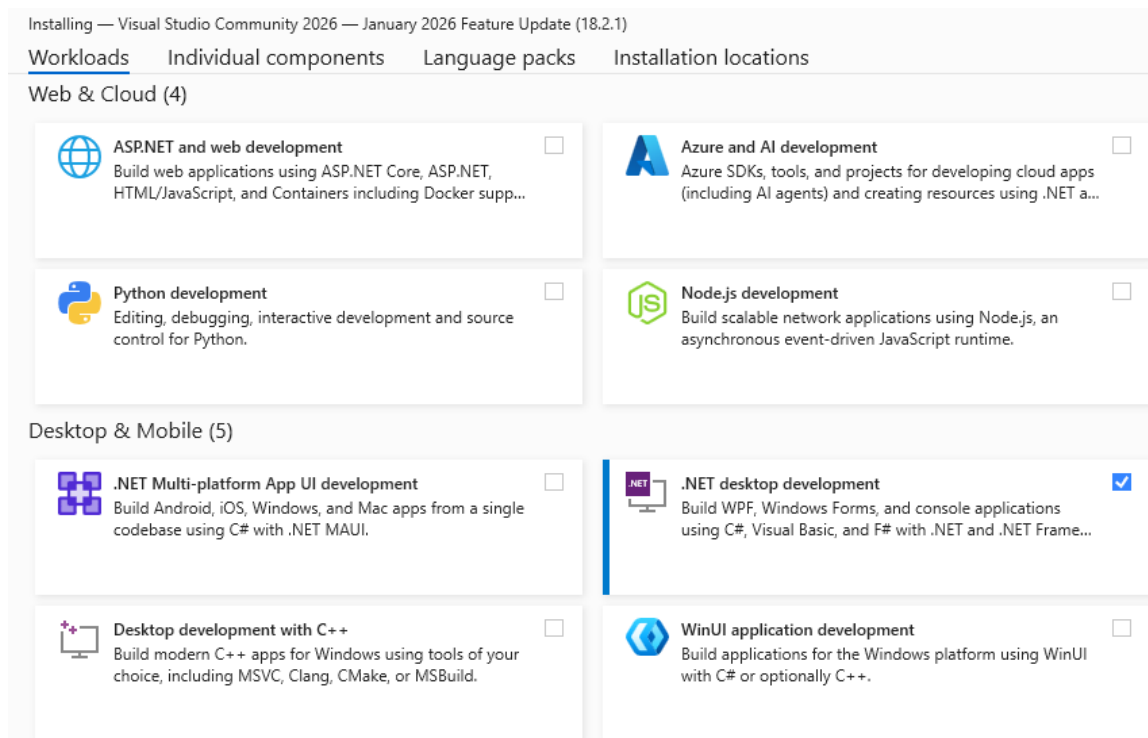
Objectives

After completing this unit you will be able to:

- **Describe Visual Studio 2026 Community and install the desired features on your computer.**
- **Use the Sign in feature of Visual Studio 2026 to synchronize settings across multiple devices.**
- **Use the Visual Studio 2026 integrated development environment (IDE) to create, edit, debug, and run C# programs.**
- **Create solutions with multiple projects.**

Visual Studio 2026 Community

- **Visual Studio 2026 Community is free and supports many features.**
- **You can choose the features you want as part of the installation.**
- **For this course we need the .NET desktop development workload.**



Modifying Visual Studio

- **After your initial installation, you may add or delete components by running the installer again.**
 - Find your installed product and click Modify.
 - You may also install updates that become available.



Visual Studio Community 2026

January 2026 Feature Update (18.2.1)

[Release notes](#)

Modify

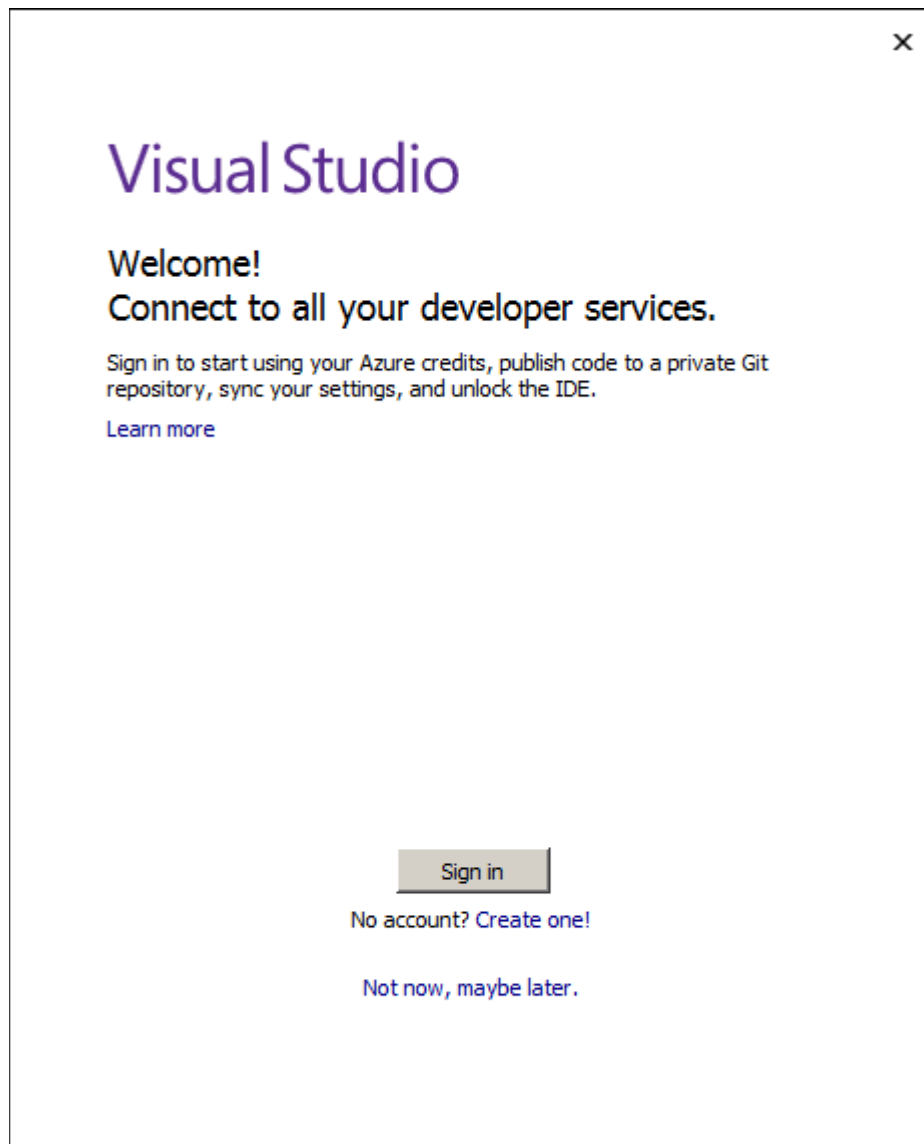
Launch

More ▼

- **After clicking Modify you can then check or uncheck to add or delete a component.**

Visual Studio Sign in

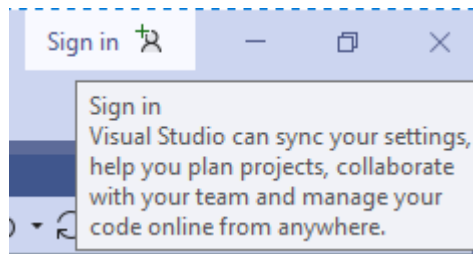
- **When you start Visual Studio 2026 for the first time you will be prompted to Sign in.**



- **If you already have a Microsoft account such as Live, Hotmail or MSDN, you can use it to sign in now.**
 - If not, you can create one for free. Use MSDN if you have it.

Sign In Advantages

- **Although it is not required, Sign In has advantages.**
 - It will automatically synchronize Visual Studio settings across devices where you run Visual Studio.
 - It will permanently unlock Visual Studio Community Edition.
 - It will extend the trial period of Visual Studio Professional or Enterprise from 30 to 90 days.
 - It will automatically unlock Visual Studio if you sign in using an MSDN account.
 - It will enable you to automatically connect to services such as Azure and a private Git repository.
- **You can sign in later. Look for the Sign in link in the upper right corner of Visual Studio:**

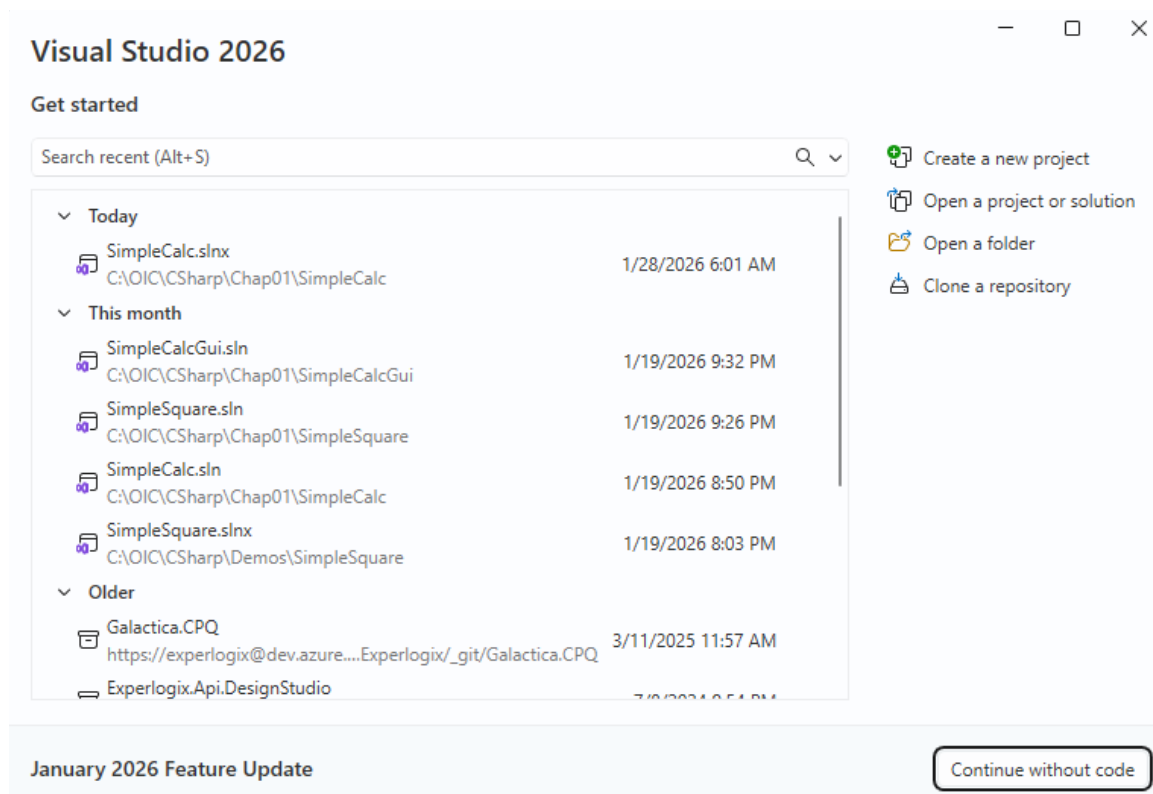


- **More information about Sign In is available on [docs.microsoft.com](https://docs.microsoft.com/en-us/visualstudio/ide/signing-in-to-visual-studio).**

<https://docs.microsoft.com/en-us/visualstudio/ide/signing-in-to-visual-studio>

Visual Studio Start Page

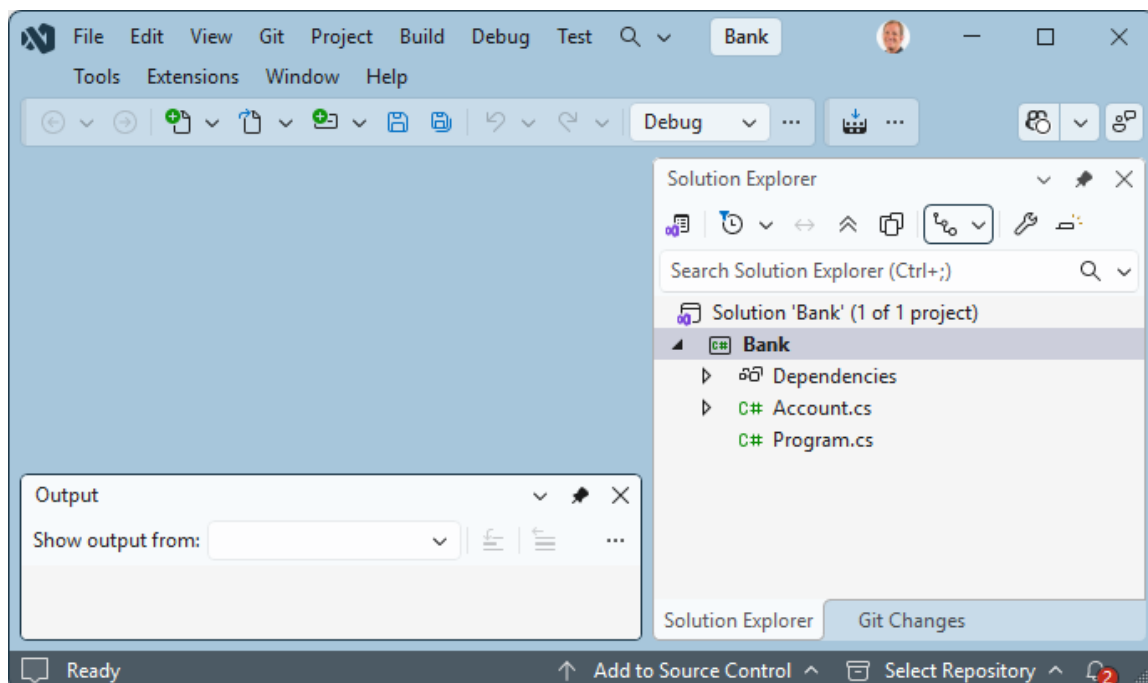
- **When you launch Visual Studio, you will see the Visual Studio Start Page.**
 - You will be given several ways of opening existing code or creating a new project.
 - A link at the bottom right lets you enter Visual Studio without any code.



- Click the Continue without code button.

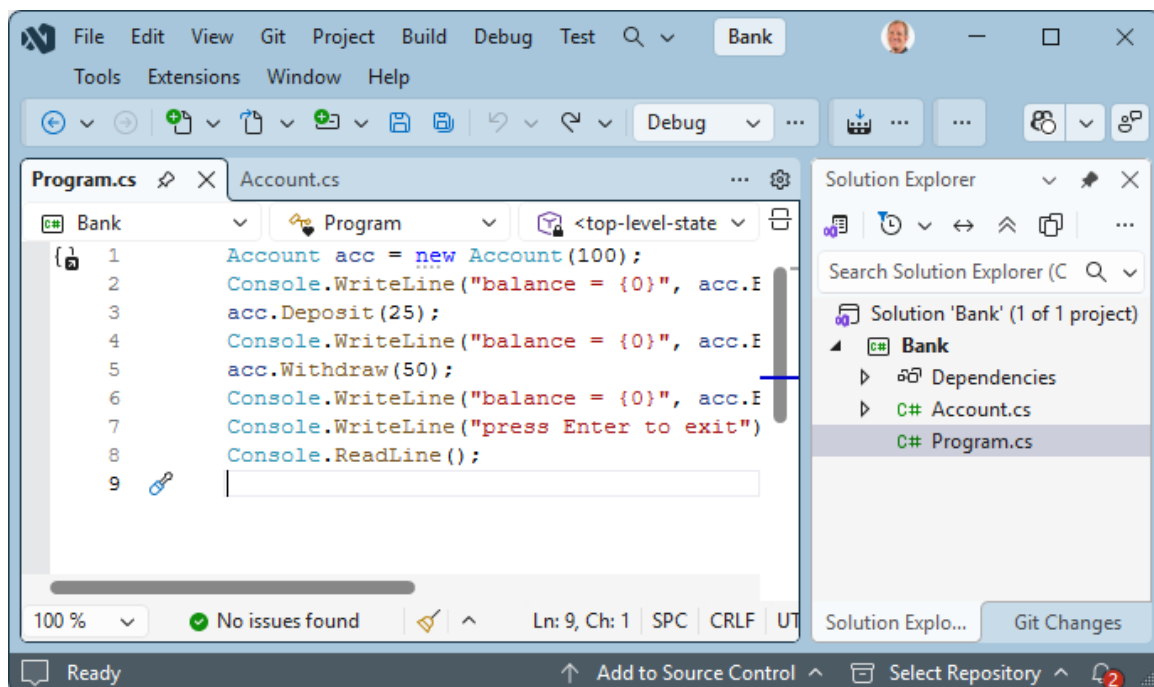
A Visual Studio Solution

- **This unit does not have a separate lab. Instead, we will work through some examples together so that you can get a good feel for how the IDE works.**
- **Open the Bank console solution.**
 - File | Open | Project/Solution.
 - Navigate to directory **Supp1\Bank**
 - Select the file **Bank.sln**, and open it.



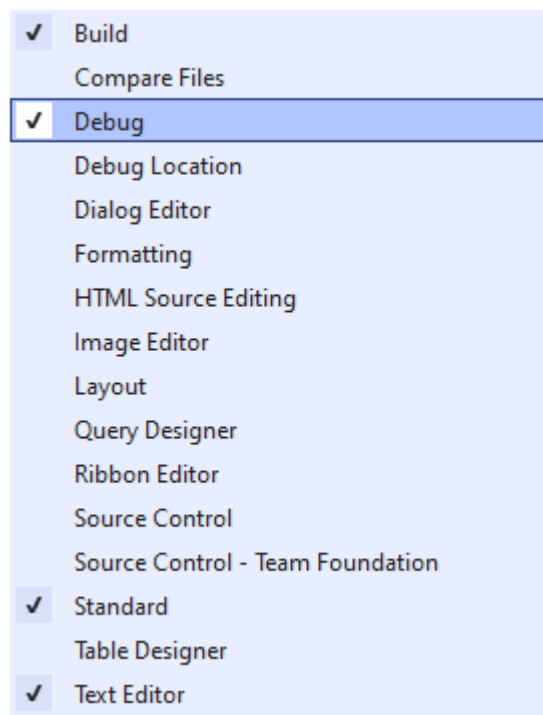
Solution Explorer

- On the top right is the Solution Explorer, which shows you the structure of your *solution*, which consists of one or more *projects*.
- Double-click on each of the files *Account.cs* and *Program.cs*, the two source files in the Bank project.
 - We closed some windows we did not need.



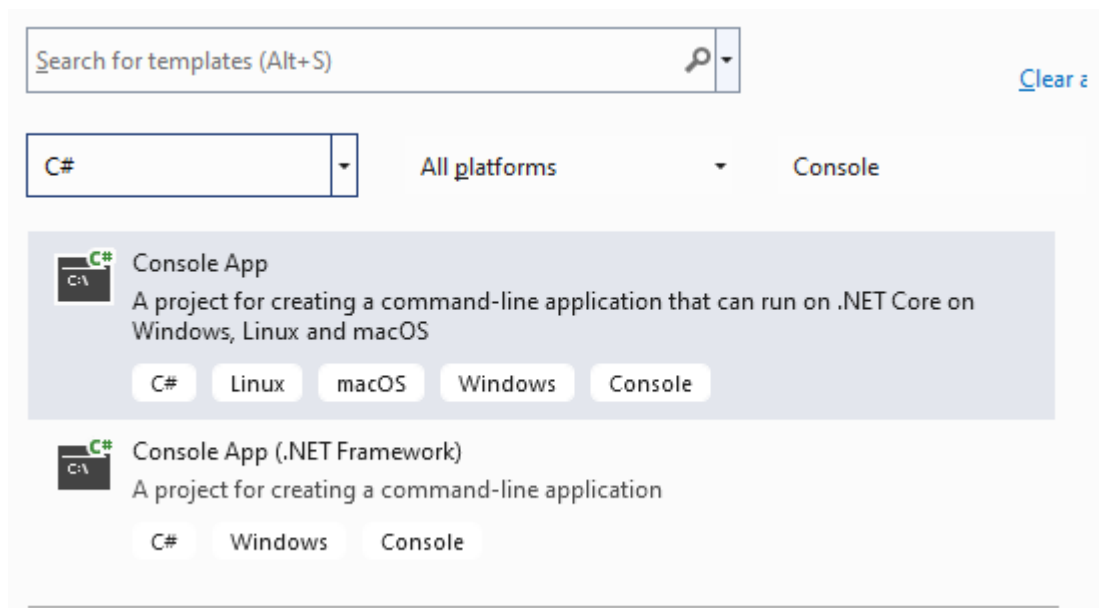
Toolbars

- **Visual Studio comes with many configurable toolbars.**
 - You can configure which toolbars are shown.
 - You can drag toolbars to whatever position you find most convenient.
 - You can add or delete buttons on the toolbars.
- **To bring up the menu specifying which toolbars are shown, choose View | Toolbars, or right-click on any empty area of a toolbar.**
- **If you don't have the Build and Debug toolbars shown, add them now.**



Creating a Console App

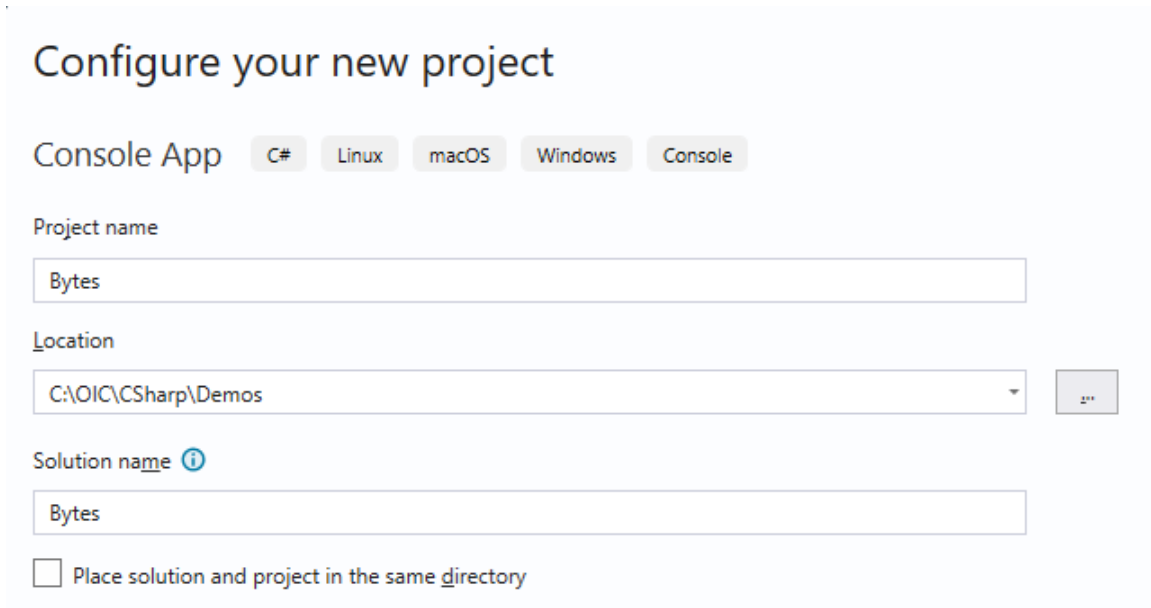
- **We will now create a simple console application.**
 - Our program **Bytes** will attempt to calculate how many bytes there are in a kilobyte, a megabyte, a gigabyte, and a terabyte.
1. If a solution is open, close it. From the main menu, choose File | New | Project.... This will bring up the New Project dialog.
 2. For Templates filter by C# and Console. Choose Console App



3. Click Next.

Creating a Console App (Cont'd)

4. For Project name, type **Bytes**. Navigate to the **Demos** directory. Leave unchecked “Place solution and project in the same directory”.



Configure your new project

Console App C# Linux macOS Windows Console

Project name

Bytes

Location

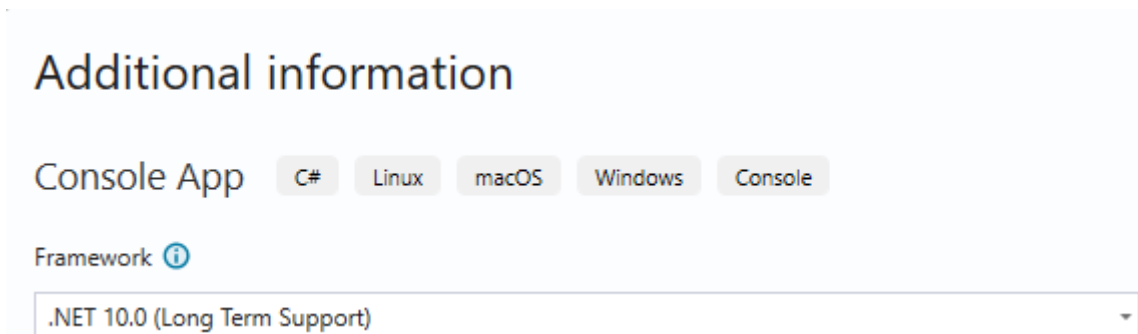
C:\OIC\CSharp\Demos

Solution name ⓘ

Bytes

☐ Place solution and project in the same directory

5. Click Next. For Framework choose .NET 10.0



Additional information

Console App C# Linux macOS Windows Console

Framework ⓘ

.NET 10.0 (Long Term Support)

6. Click Create.




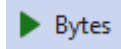
Using the Visual Studio Text Editor

- The empty code file *Bytes.cs* will open.
- In *Program.cs* enter the following code using the Visual Studio text editor:

```
int bytes = 1024;  
Console.WriteLine("kilo = {0}", bytes);  
bytes = bytes * 1024;  
Console.WriteLine("mega = {0}", bytes);  
bytes = bytes * 1024;  
Console.WriteLine("giga = {0}", bytes);  
bytes = bytes * 1024;  
Console.WriteLine("tera = {0}", bytes);
```

- Notice that the Visual Studio text editor highlights.
- As you type, little IntelliSense boxes pop up for code completion.

Build and Run the Bytes Project

- **You can build the project by using one of the following:**
 - Menu Build | Build Solution or toolbar button  or keyboard shortcut Ctrl+Shift+B.
 - Menu Build | Build Bytes or toolbar button  (this just builds the project Bytes)¹.
- **You can run the program without debugging by using one of the following:**
 - Menu Debug | Start Without Debugging
 - Toolbar button 
 - Keyboard shortcut Ctrl + F5
- **You can run the program in the debugger by using one of the following:**
 - Menu Debug | Start Debugging
 - Toolbar button 
 - Keyboard shortcut F5
- **Try it!**

¹ The two are the same in this case, because the solution has only one project, but some solutions have multiple projects, and then there is a difference. We'll create a solution with two projects later.

Running the Bytes Project

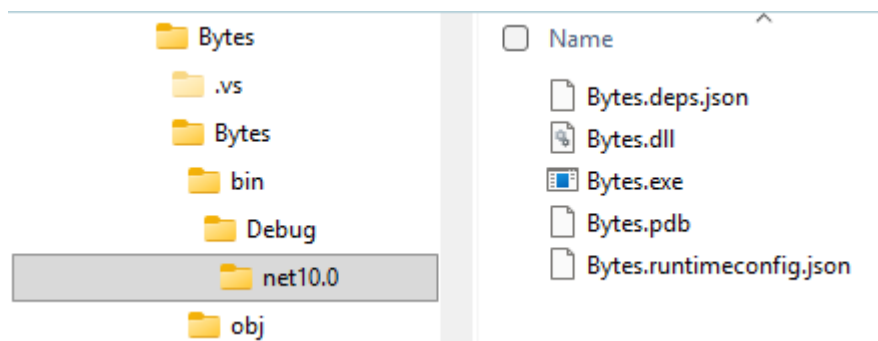
- **You will get some obviously erroneous output!**

```
kilo = 1024  
mega = 1048576  
giga = 1073741824  
tera = 0
```

- The solution is saved at this point in **Bytes\Step1**.

Executable File Location

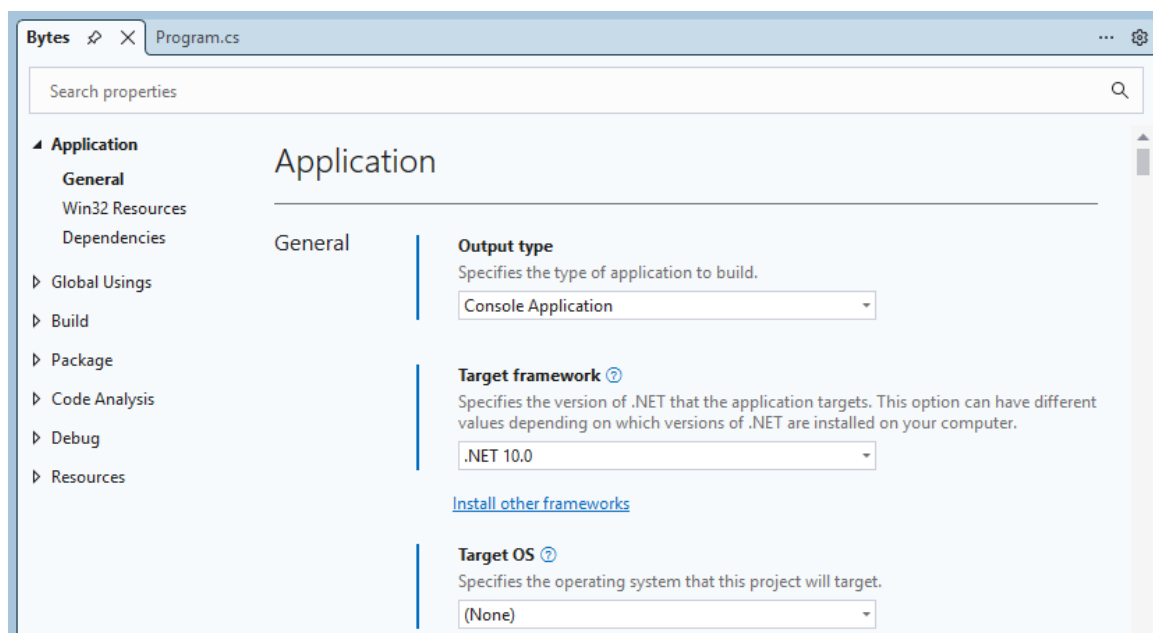
- When you build a project from Visual Studio, by default the output files are placed under the *bin\Debug* directory.
 - They will be in a subdirectory corresponding to the version of the .NET Framework.



- The executable file for the **Bytes** program is **Bytes.exe**.
- When you build the program with a Release configuration, the executable file is placed under the *bin\Release* directory.

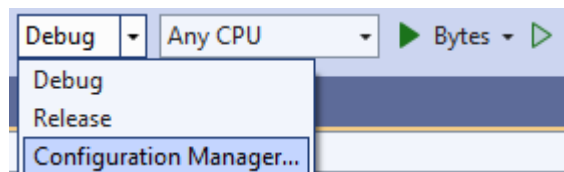
Project Designer

- **You may open the Project Designer by right-clicking on the project in Solution Explorer and choosing Properties.**
 - This allows you to specify properties of the project, for example the Target Framework.

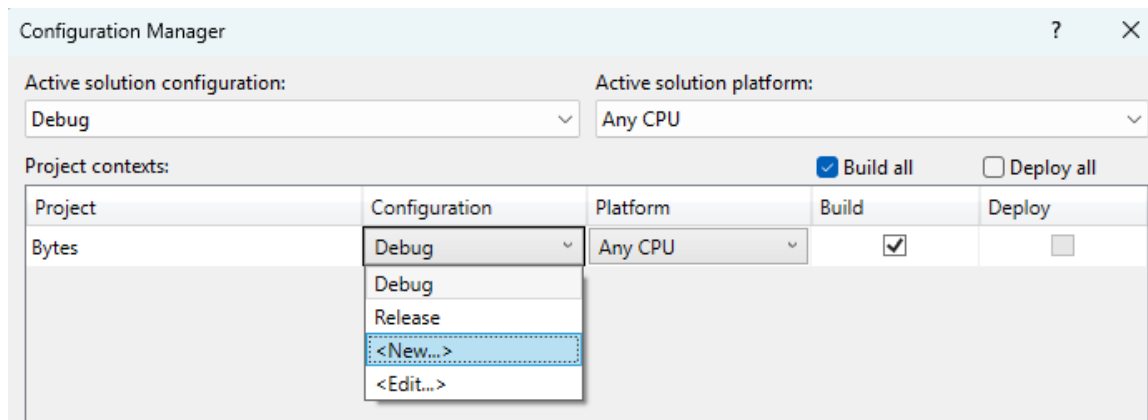


Project Configurations

- A project *configuration* specifies build settings for a project.
- Every project in a Visual Studio solution has two default configurations, *Debug* and *Release*.
- A dropdown on the standard toolbar lets you select a configuration for the build.

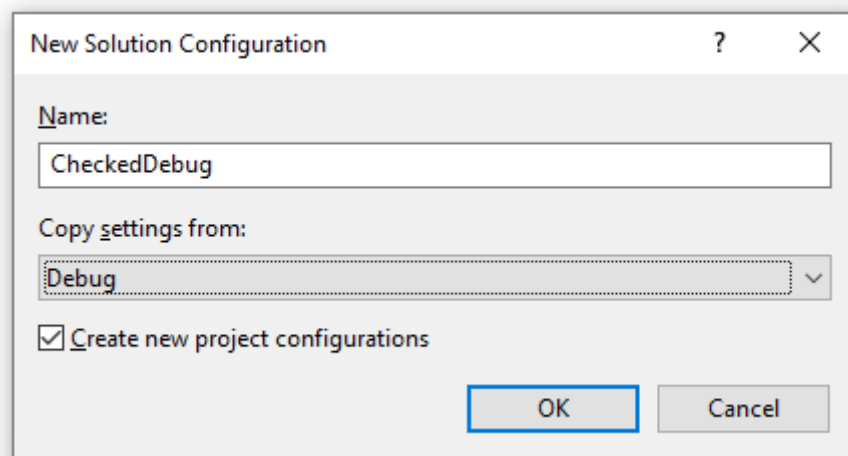


- The Configuration Manager gives you complete control over the configuration, including creating a new (custom) configuration.



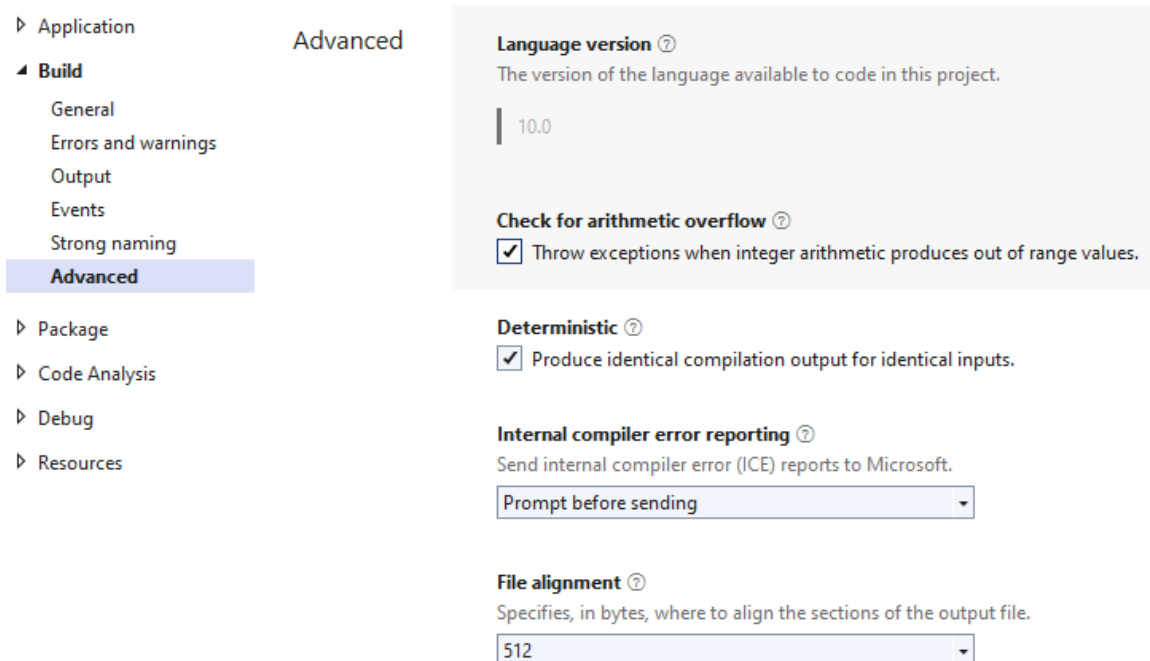
Creating a New Configuration

- Sometimes it is useful to create additional configurations, which can save alternate build settings.
 - As an example, let's create a configuration for a “checked” build.
1. Bring up the Configuration Manager dialog.
 2. From the Active Solution Configuration: dropdown, choose <New...>. The New Solution Configuration dialog will come up.
 3. Type **CheckedDebug** as the configuration name. Choose Copy Settings from **Debug**. Leave checked “Create new project configurations” (see below). Click OK, and then Close from the Configuration Manager dialog.



Setting Configuration Build Settings

- **Next, we will set the build settings for the new configuration.**
 - Inspect the toolbar to verify that the new **CheckedDebug** is the currently-active configuration.
1. Right-click over **Bytes** in the Solution Explorer and choose Properties. The Project Designer comes up.
 2. Select Advanced under Build from the list at the left.
 3. Check “Check for arithmetic overflow”.



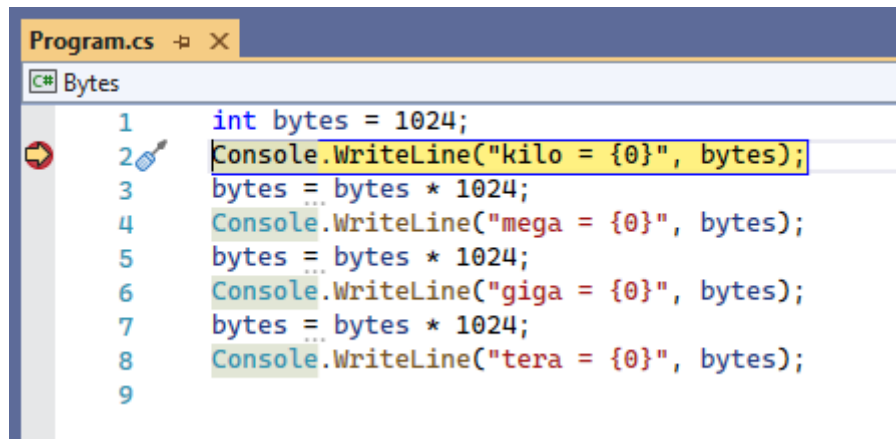
4. Click OK.
5. Build and run. Now you will hit an Overflow exception. The solution is saved at this point in **Bytes\Step2**.

Debugging

- **To be able to benefit from debugging at the source code level, you should have built your executable using a Debug configuration, as discussed previously.**
- **There are two ways to enter the debugger:**
 - **Standard Debugging.** You start the program under the debugger. You may set breakpoints, single step, and so on.
 - **Just-in-Time Debugging.** You run normally and, if an exception occurs, you will be allowed to enter the debugger. The program has crashed, so you will not be able to run further from here to single step, set breakpoints, and so on. But you will be able to see the value of variables, and you will see the point at which the program failed.
- **We will only use standard debugging in this course.**

Breakpoints

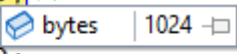
- **The way you typically perform standard debugging is to set a breakpoint and then run using the debugger.**
 - The easiest way to set a breakpoint is by clicking in the gray bar to the left of the source code window. You will then see a red dot.
 - Now start with debugging. A yellow arrow over the red dot of the breakpoint shows where the breakpoint has been hit.



Watch Variables


- The easiest way to inspect the value of a variable is to slide the mouse over the variable you are interested in and the value will be shown as a tool tip.

```
int bytes = 1024;  
Console.WriteLine("kilo = {0}", bytes);  
bytes = bytes * 1024;  
Console.WriteLine("mega = {0}", bytes);
```









- Click the pushpin, and the little window stays open, and you can watch the value change as you step through the program.

```
int bytes = 1024;  
Console.WriteLine("kilo = {0}", bytes);  
bytes = bytes * 1024;  
Console.WriteLine("mega = {0}", bytes);  
bytes = bytes * 1024;  
Console.WriteLine("giga = {0}", bytes);  
bytes = bytes * 1024;  
Console.WriteLine("tera = {0}", bytes);
```



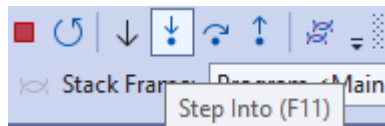
≤ 1ms elapsed

- You can display a variable in a Watch window.
- This illustration shows a typical Watch window. You can show the Watch window through the tab at the bottom.

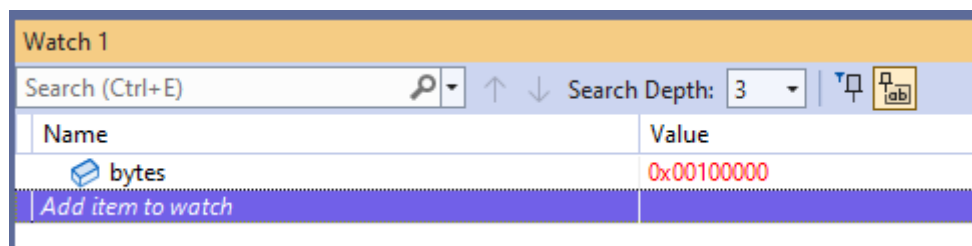
| Watch 1 | |
|---|---------|
| Search (Ctrl+E)    Search Depth: 3   | |
| Name | Value |
|  bytes | 1048576 |
| Add item to watch | |

Debug Toolbar

- **The Debug toolbar gives quick access to several useful debugging commands.**
 - Slide the mouse cursor over the buttons to see a yellow tooltip giving a brief description.




- **You can get a hexadecimal display by right-clicking over a variable in the Watch window and selecting Hexadecimal display.**
- **You will see the value in the Watch window now displayed in hex.**





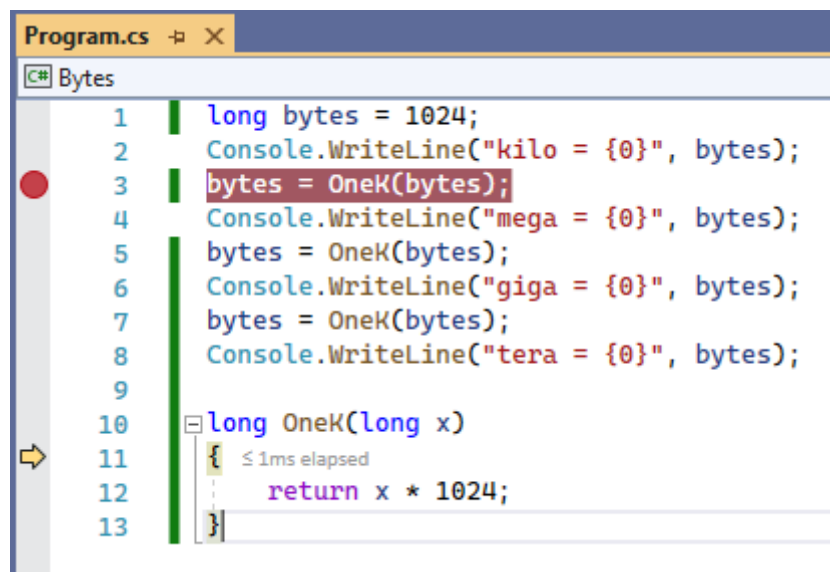
- The red coloring indicates that in this run the value has just changed.

Stepping with the Debugger

- When you are stopped in the debugger, you can step through instructions either one at a time or by set amounts.
- There are several single step buttons . The default ones are (in the order shown on the toolbar):
 - Step Into
 - Step Over
 - Step Out
- There are additional buttons that can be added to the Debug toolbar via **Tools | Customize**.

Demo: Stepping with the Debugger

- Build the *Bytes\Step3* project.
 - The multiplication by 1024 has been replaced by a function. We're also using the **long** data type to fix the bug.
- Set a breakpoint at the first function call, start the program ( Bytes), and then Step Into ().
 - Note the red dot at the breakpoint and the yellow arrow in the function.



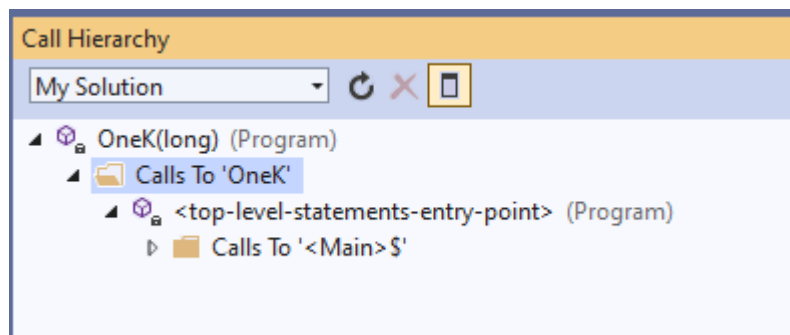
```
Program.cs [X]
C# Bytes
1  long bytes = 1024;
2  Console.WriteLine("kilo = {0}", bytes);
3  bytes = OneK(bytes);
4  Console.WriteLine("mega = {0}", bytes);
5  bytes = OneK(bytes);
6  Console.WriteLine("giga = {0}", bytes);
7  bytes = OneK(bytes);
8  Console.WriteLine("tera = {0}", bytes);
9
10 long OneK(long x)
11 {
12     return x * 1024;
13 }
```

Call Stack and Call Hierarchy

- When debugging, Visual Studio maintains a **Call Stack**.
- In our simple example, the **Call Stack** is just two deep.

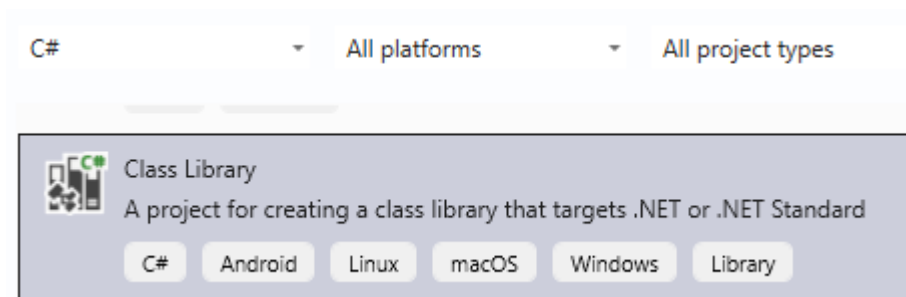


- If you right-click on a method name, you can select **View Call Hierarchy**.



Multiple-Project Solution Demo

- **Visual Studio solutions may contain multiple projects.**
 - This feature is useful in organizing larger programs.
 - **As an example, let's add a second project to our bytes solution.**
 - Starter code is provided in **Demos\Multiple**, which is a copy of **Bytes\Step4**. This version provides the method **OneK()** as a static method in the class **SimpleMath** in a separate file.
1. Examine the starter code. Build and run.
 2. In Solution Explorer, right-click over the solution and choose Add | New Project... from the context menu.
 3. Select the Class Library.



4. Click Next

Multiple-Project Demo (Cont'd)

5. Enter the name **MyMath** for the new project.

Configure your new project

Class Library

C#

Android

Linux

macOS

Windows

Library

Project name

MyMath

Location

C:\OIC\CSharp\Demos\Multiple

Project will be created in "C:\OIC\CSharp\Demos\Multiple\MyMath\"

6. Click Next. Select .NET 10.0 as the Framework. Click Create.

Class Library

C#

Android

Linux

macOS

Windows

Library

Framework ⓘ

.NET 10.0 (Long Term Support)

7. In the new project, delete the file **Class1.cs**.

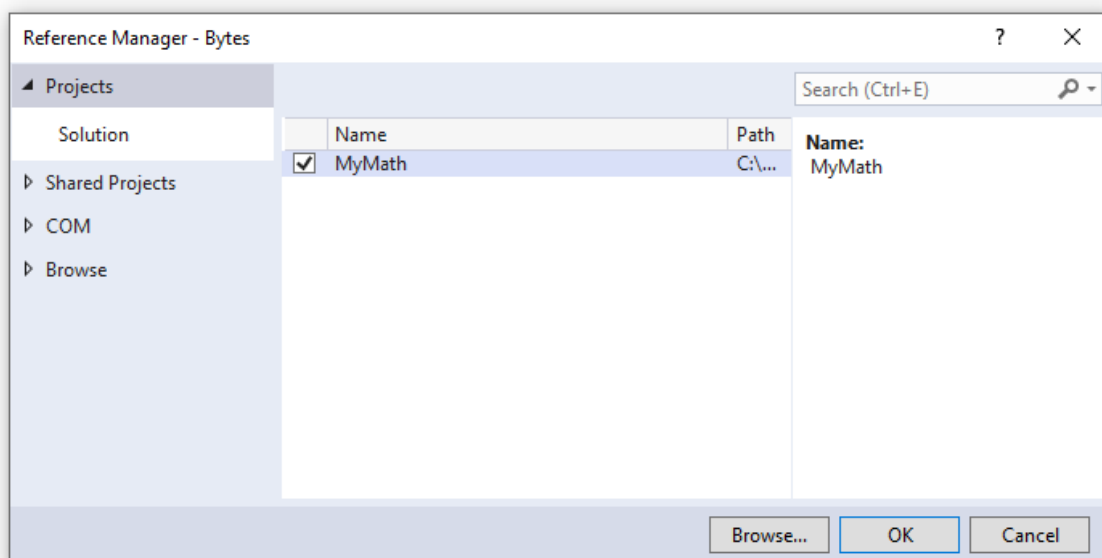
8. Drag the file **SimpleMath.cs** from the **Bytes** project to the **MyMath** project. Delete the file from the **Bytes** project.

9. Build the solution. You will get an error message.

The name 'SimpleMath' does not exist in the current context

Adding a Reference

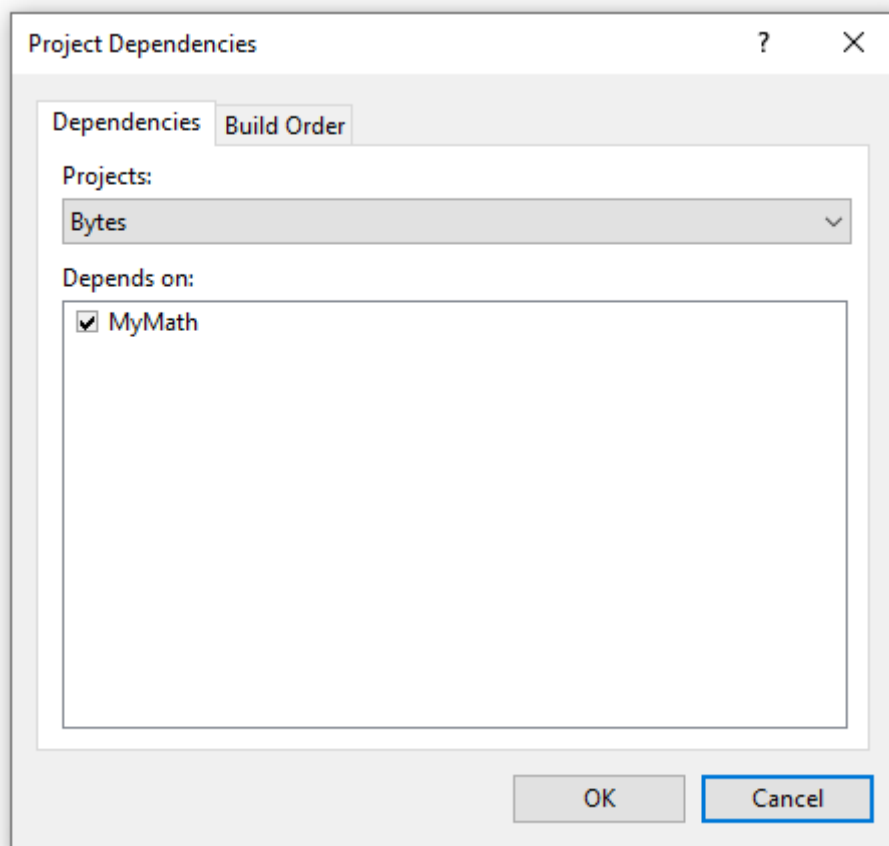
10. Right-click over the Bytes project and choose Add | Project Reference from the context menu.
11. The Reference Manager dialog comes up. Select Projects. There is only one other project in this solution, **MyMath**. Select it.



12. Click OK.
13. Build and run. You now will get a different error:
`'SimpleMath' is inaccessible due to its protection level`
14. Make the class **SimpleMath** public. Now it should work!
Solution is saved in **Supp1\Multiple**.

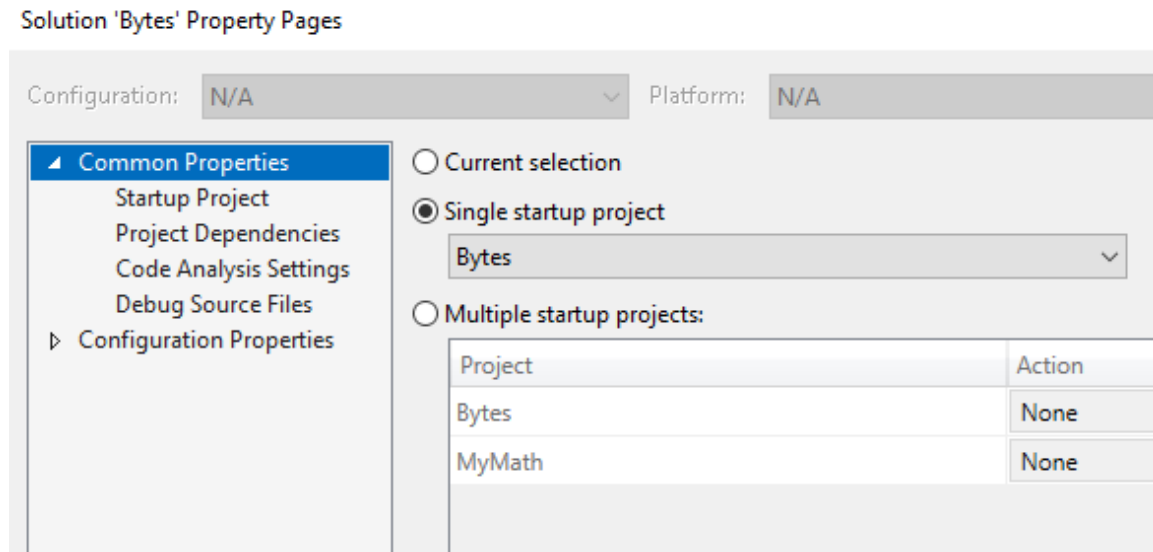
Project Dependencies

- **Project dependencies have been set so that the class library is built first.**
 - Right-click on the solution and select Project Dependencies from the context menu. Choose the Bytes project. It depends on the MyMath project, so the latter will be built first.



Startup Project

- In a solution with multiple projects, you can choose the Setup Project from the solution properties.



- The startup project information is saved in the *.suo* file, which can be found nested under the *.vs* folder.

Summary

- **Visual Studio 2026 is a very rich integrated development environment (IDE), with many features to make programming more enjoyable.**
- **You can use the Sign in feature of Visual Studio 2026 to synchronize settings across multiple devices.**
- **In this unit, we covered the basics of using Visual Studio to edit, compile, run, and debug programs, so that you will be equipped to use Visual Studio in the rest of the course.**
- **A project can be built in different configurations, such as Debug and Release,**
- **In this course, we will use only a tiny fraction of the capabilities of this powerful tool.**
- **Multiple-project solutions are useful for larger programs.**

