

Supplement 1

Using Visual Studio 2015

Using Visual Studio 2015

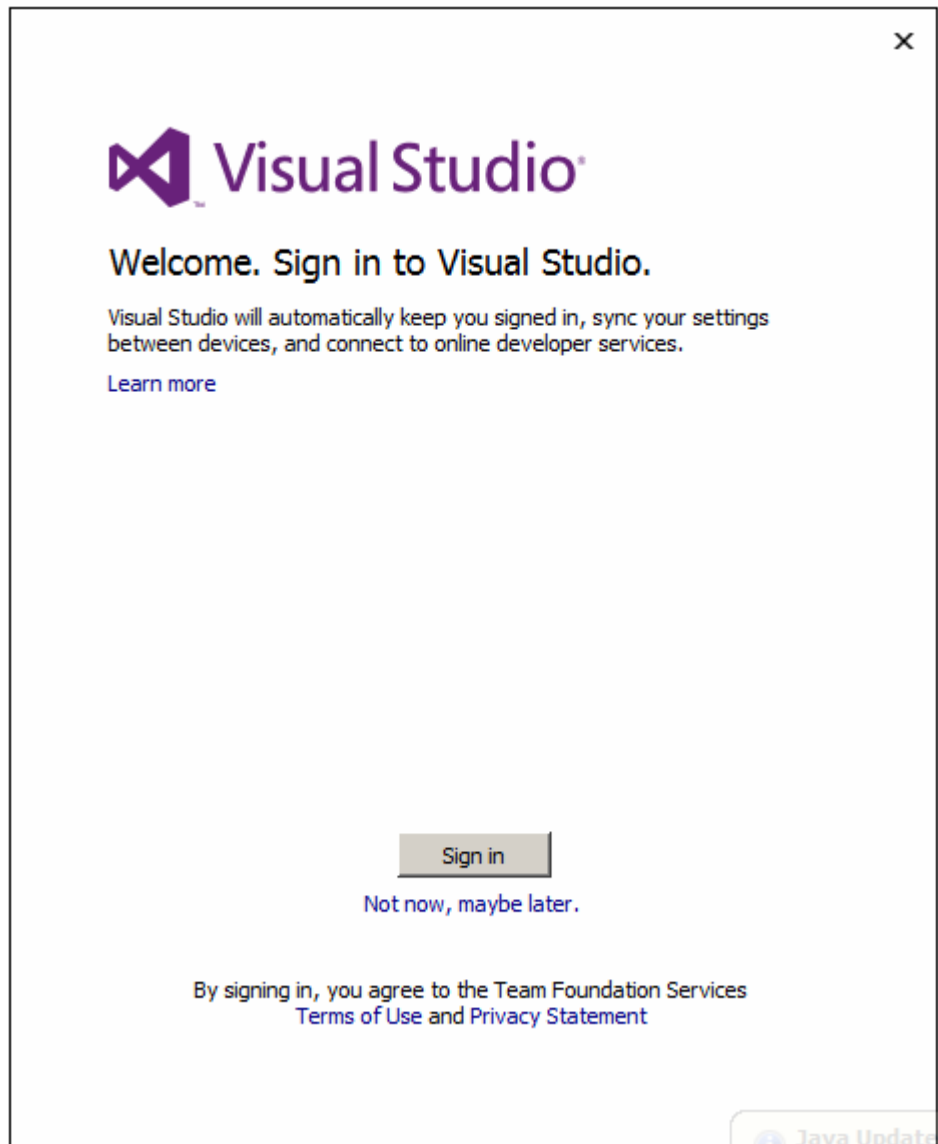
Objectives

After completing this unit you will be able to:

- **Use the Sign in feature of Visual Studio 2015 to synchronize settings across multiple devices.**
- **Use the Visual Studio 2015 integrated development environment (IDE) to create, edit, debug, and run C# programs.**
- **Create solutions with multiple projects.**

Visual Studio Sign in

- **When you start Visual Studio 2015 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.

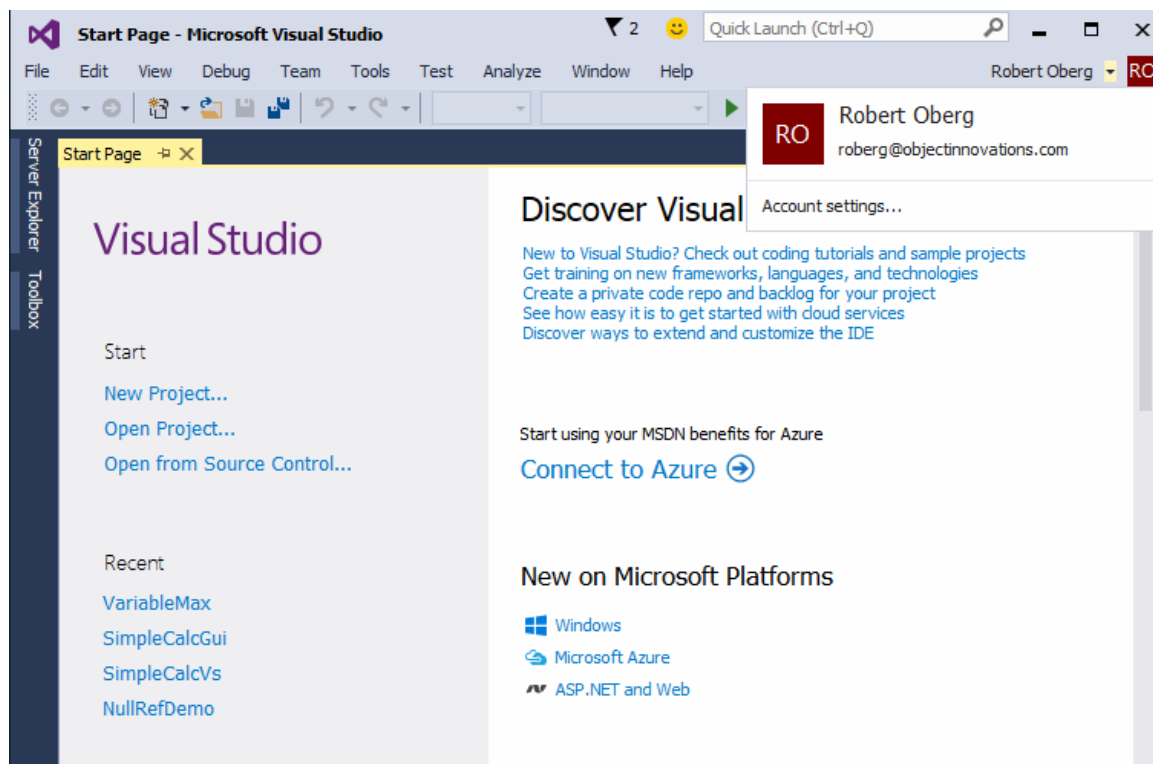
Sign In Advantages

- **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, not limiting you to a 30-day trial period.
 - 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 Visual Studio Online.
- **More information about Sign In is available on MSDN.**

[https://msdn.microsoft.com/en-us/library/vstudio/dn457348\(v=vs.140\).aspx](https://msdn.microsoft.com/en-us/library/vstudio/dn457348(v=vs.140).aspx).

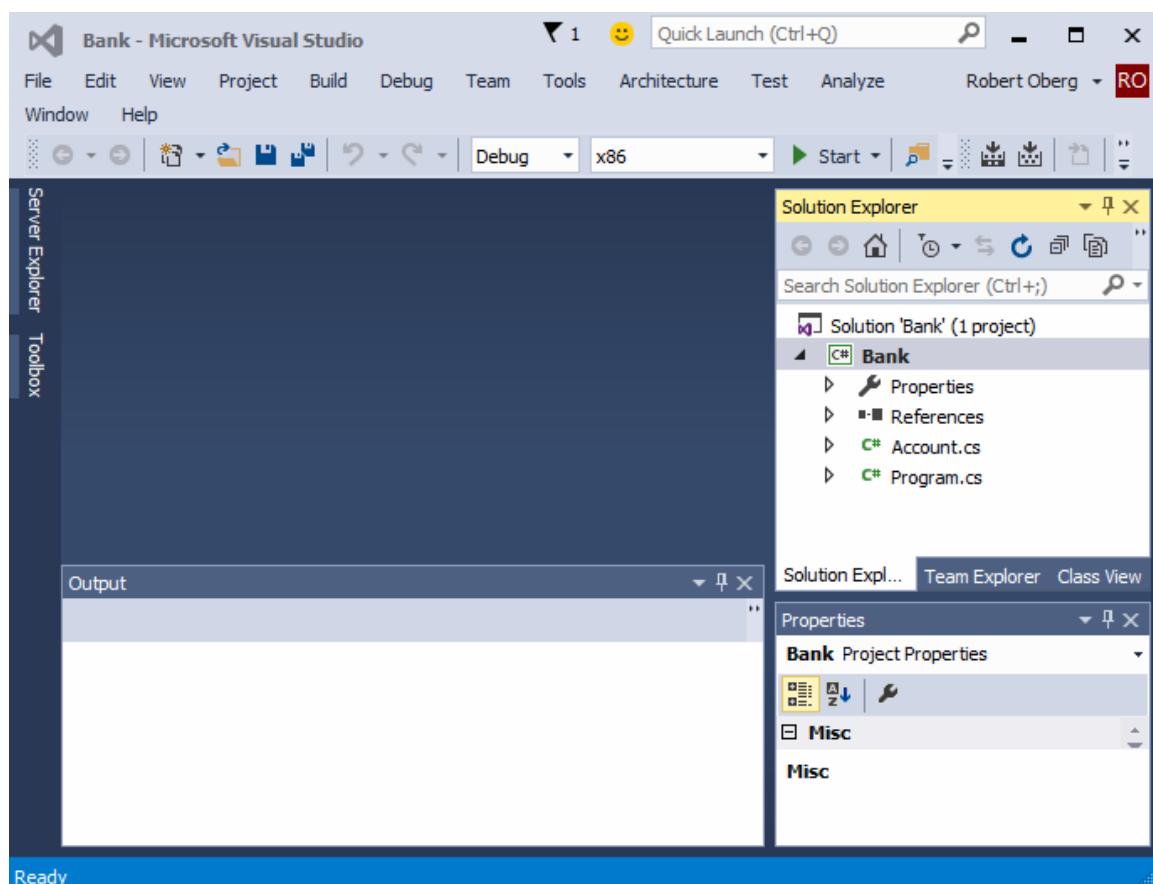
Visual Studio Start Page

- **After you have signed in you will see the Visual Studio Start Page.**
 - At top right you will see your name and icon for your Microsoft account.
 - A dropdown gives you access to your account settings.



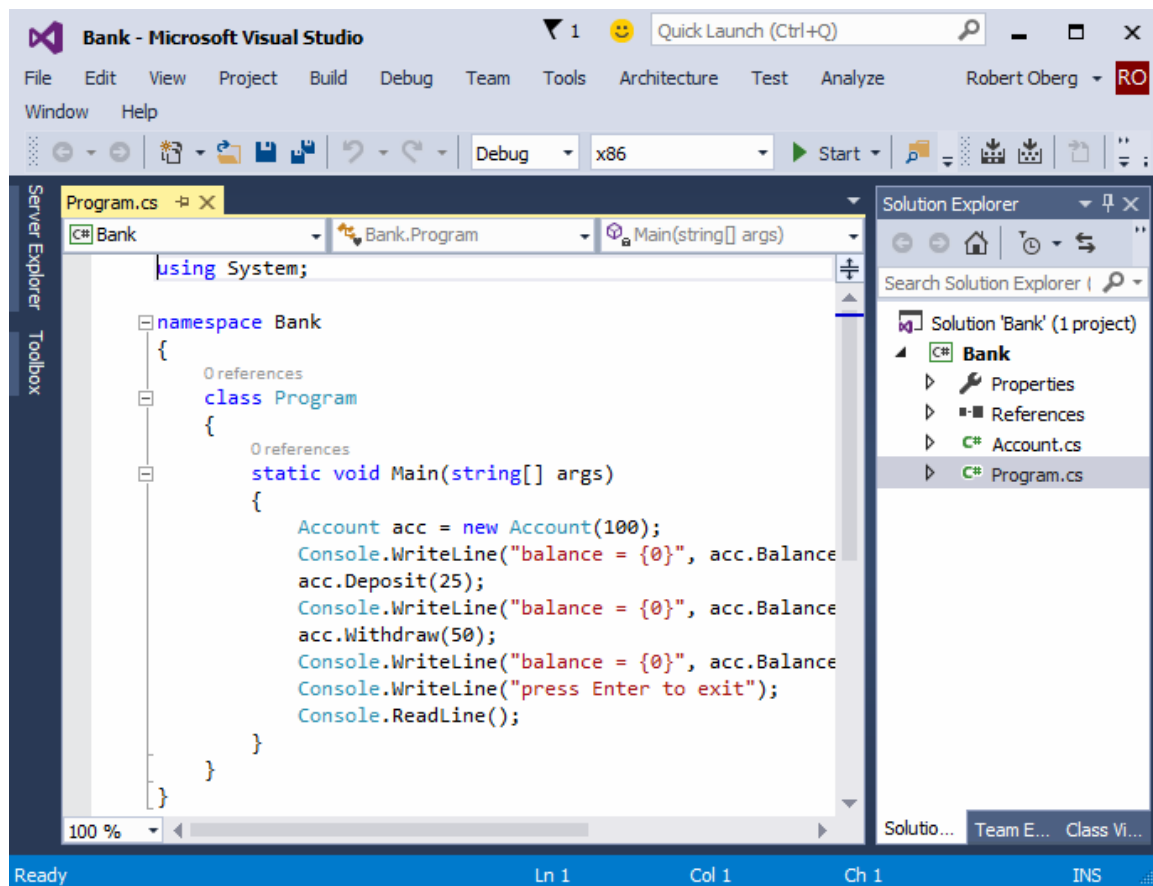
A Visual Studio Solution

- 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. (The Start Page disappears when a project is opened.)
 - Select the Bank project in the solution and expand 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 *Bank.cs*, the two source files in the Bank project.
 - We closed some windows we did not need.



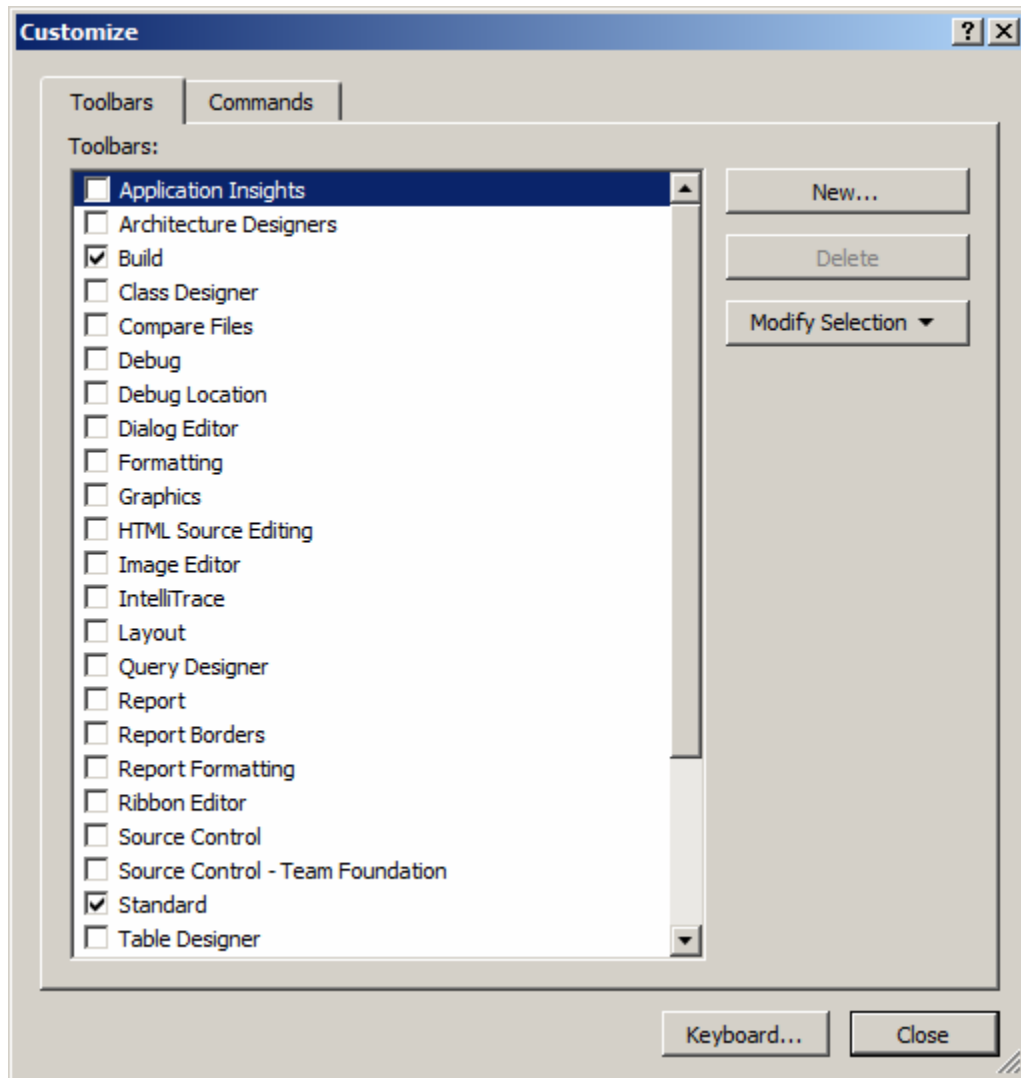
Toolbars

- **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.**

Customizing a Toolbar

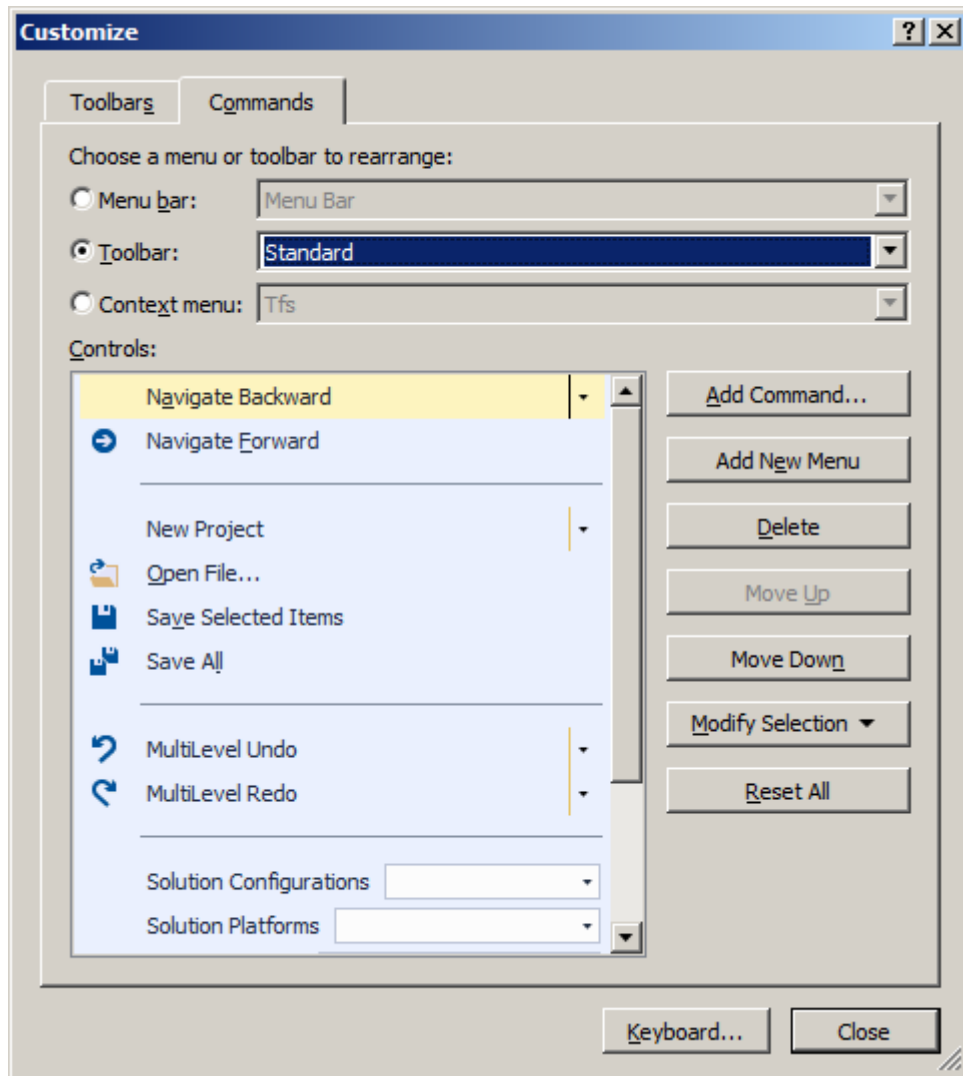
- We will add the "Start Without Debugging" button  to the Standard toolbar.

1. Select menu Tools | Customize... to bring up the Customize dialog.



Customizing a Toolbar (Cont'd)

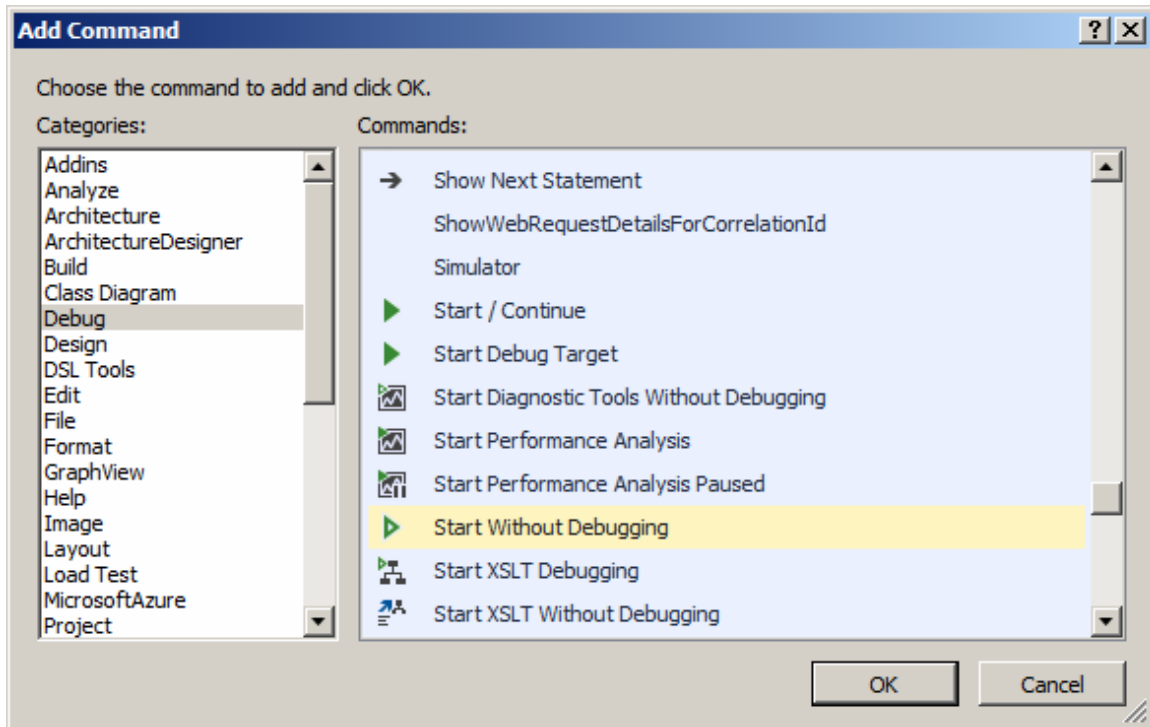
2. Select the Commands tab. Choose the Toolbar radio button and Standard from the dropdown.




3. Click the Add Command... button.

Customizing a Toolbar (Cont'd)

4. In Categories, select Debug. In Commands, scroll down and select Start Without Debugging.

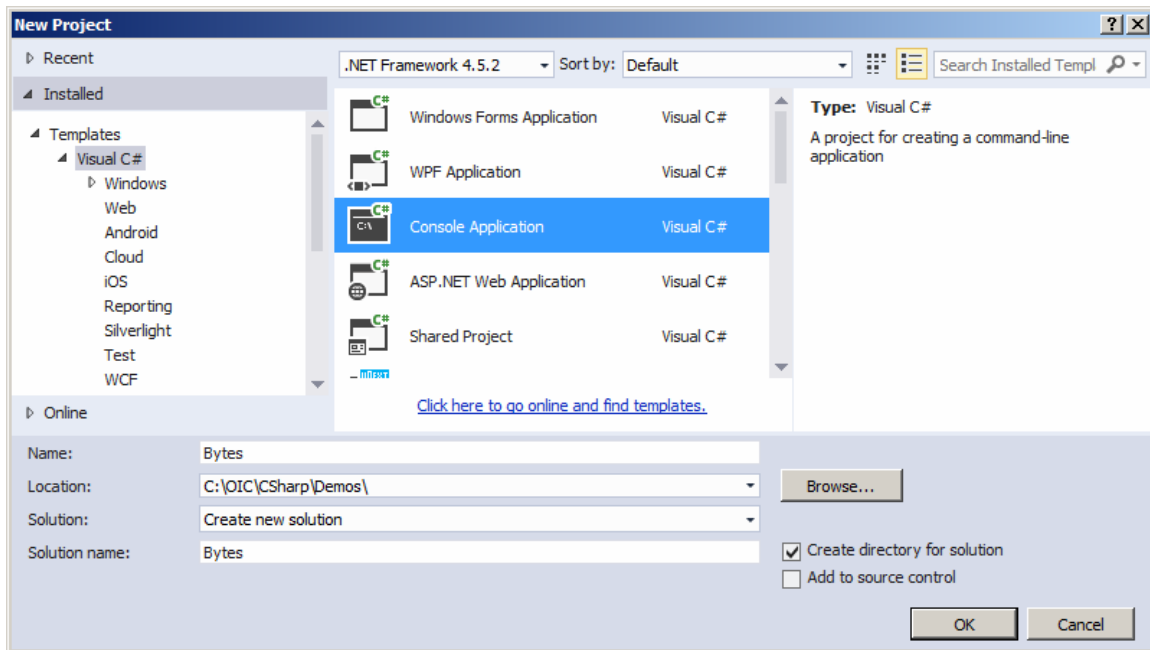


5. Click OK and then Close.
6. The new button will be in the Standard commands. Click the Move Down or Move Up button if necessary to position the new button where you want it. You can see the button move on the toolbar.
7. Close the Customize dialog.
8. Observe that the button  is now part of the Standard toolbar.



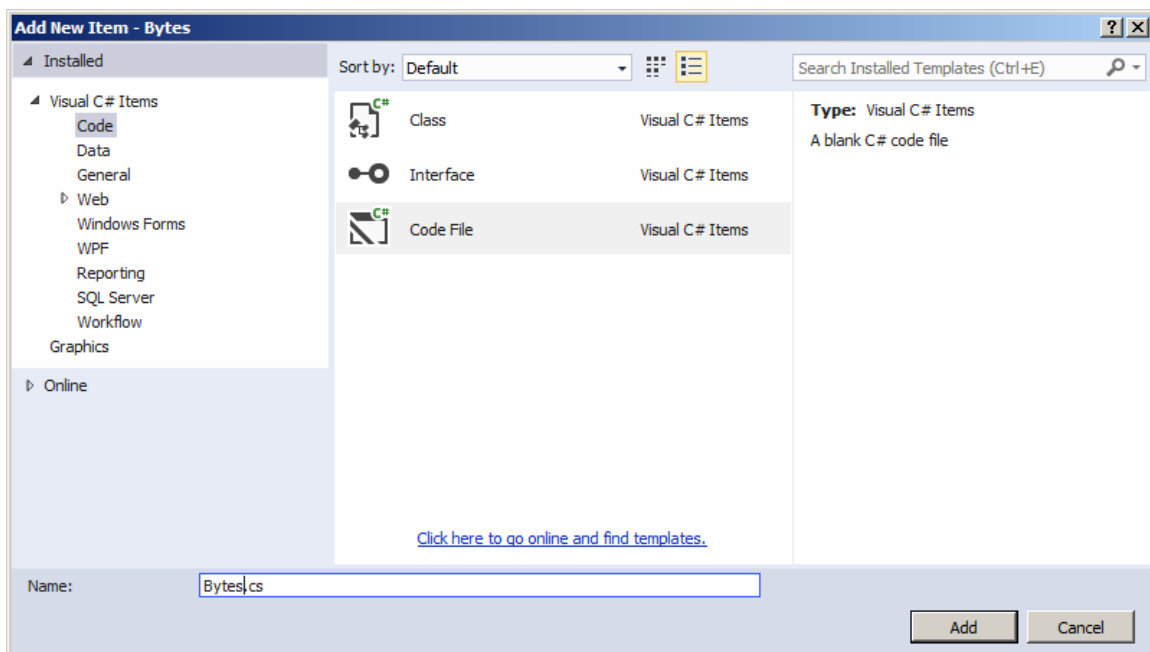
Creating a Console Application

- 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 choose Console Application from the Visual C# group.
 3. Navigate to the **Demos** directory. In the Name field, type **Bytes**. Leave “Create directory for solution” checked. Click OK.



Adding a C# File

- There will be a number of starter files. Expand properties and select the files *AssemblyInfo.cs*, *App.config* and *Program.cs*. Press the Delete key.
 - We are now going to add a file *Bytes.cs*, which contains the text of our program.
1. In Solution Explorer, right click over **Bytes** and choose Add | New Item.... This will bring up the Add New Item dialog.
 2. In the middle pane, choose “Code File.” For Name type **Bytes.cs**. Click Add.



Using the Visual Studio Text Editor




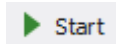
- The empty code file *Bytes.cs* will open.
- Enter the following program using the Visual Studio text editor:

```
using System;

class Bytes
{
    static void Main(string[] args)
    {
        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 syntax and indents automatically.
- 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  (which we added as a customization)
 - Keyboard shortcut Ctrl + F5
- **You can run the program in the debugger by using one of the following:**
 - Menu Debug | Start Debugging
 - Toolbar button  Start
 - 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

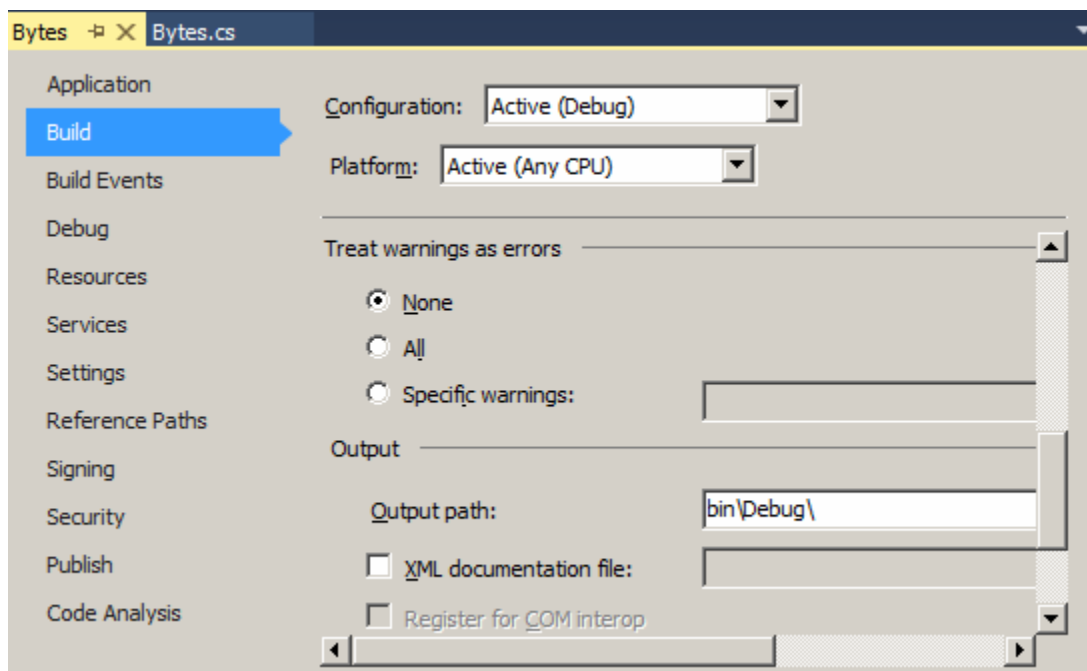
- **You will get some obviously erroneous output!**

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

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

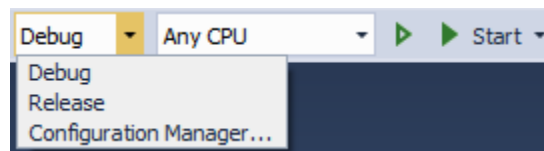
Executable File Location

- When you build a project from Visual Studio, by default the output file is placed in the *bin\Debug* directory.
 - The executable file for the **Bytes** program is **Bytes.exe**.
- When you build the program with a Release configuration, the executable file is placed in the *bin\Release* directory.
- You may change the output path by using the Project Designer, which is opened by right-clicking on the project in Solution Explorer and choosing Properties.
 - Select Build tab on left and scroll down.

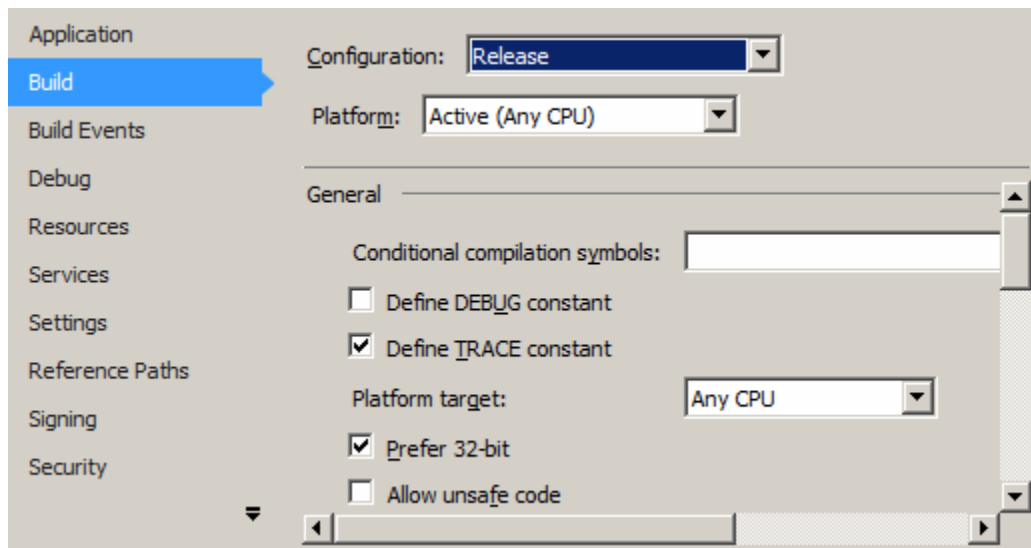


Managing Configurations

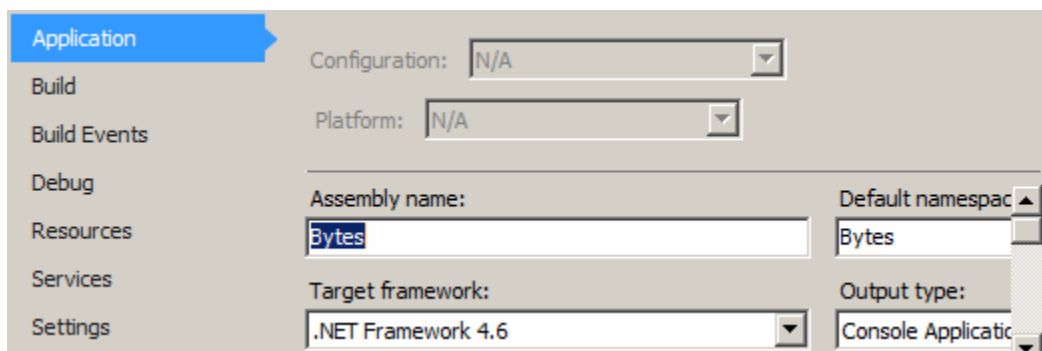
- A dropdown on the standard toolbar lets you select a configuration for the build.



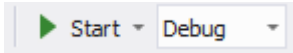
- You can also specify a configuration in the Project Designer.

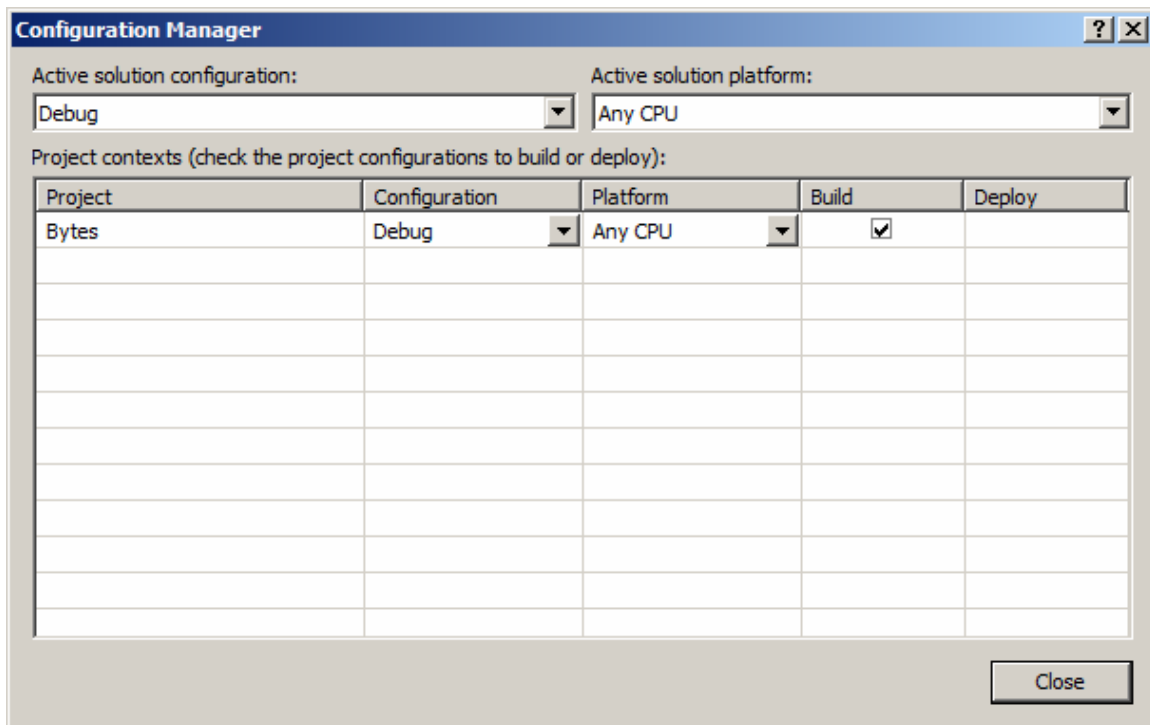


- Another setting (in the Application tab) specifies the version of the .NET Framework that will be used.



Project Configurations

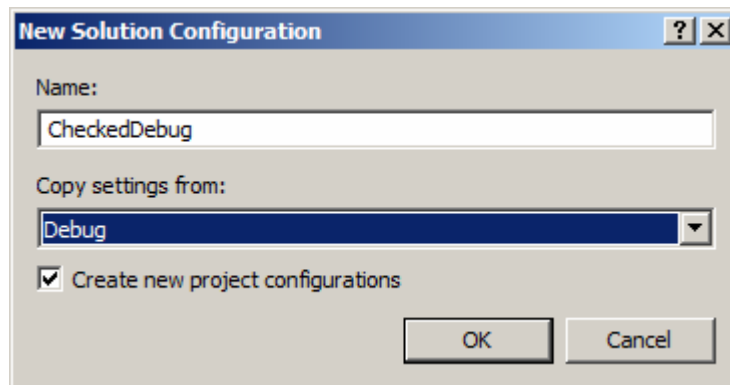
- A project *configuration* specifies build settings for a project.
- Every project in a Visual Studio solution has two default configurations, *Debug* and *Release*.
- You can choose the configuration from the main toolbar  or using the menu Build | Configuration Manager..., which will bring up the Configuration Manager dialog.



- You can also use the Project Designer, as illustrated on the previous page.

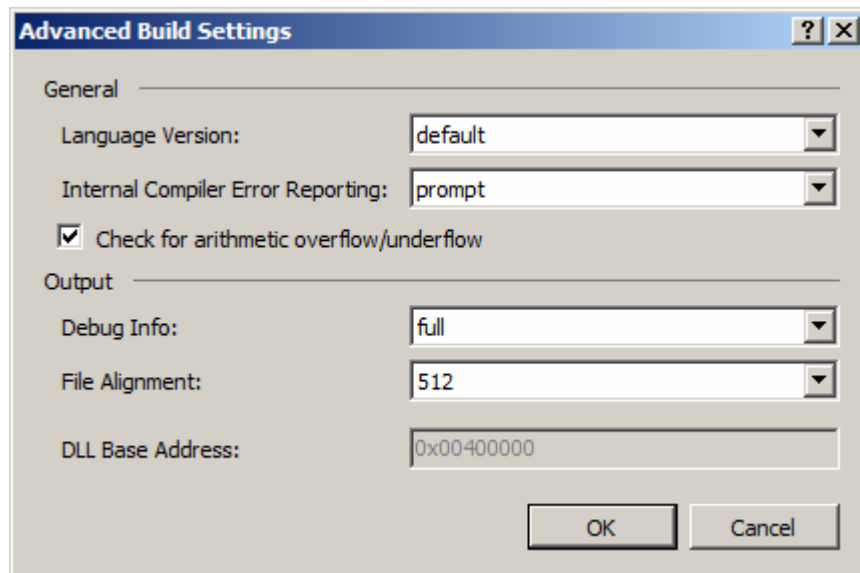
Creating a New Configuration

- 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

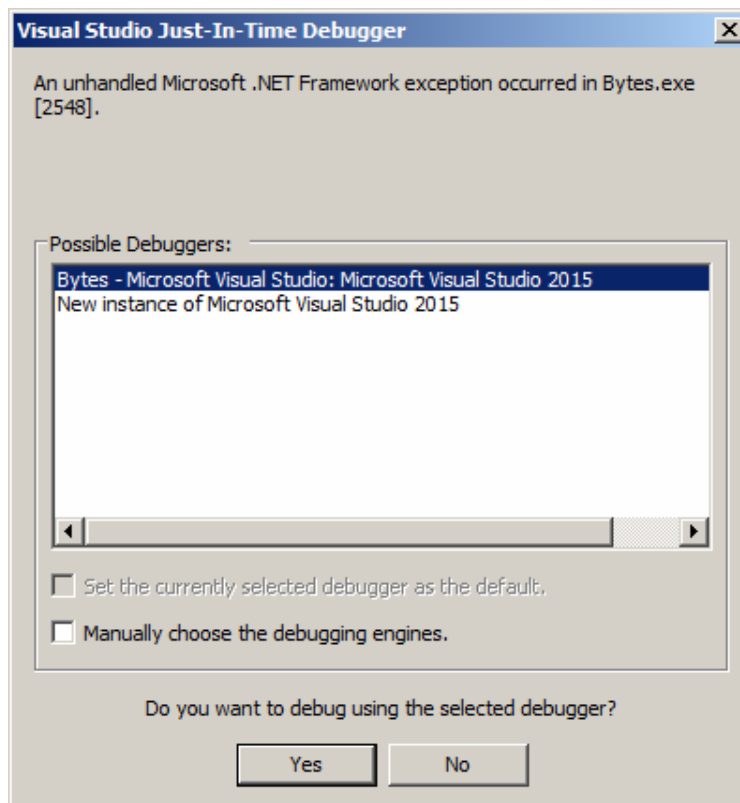
- **Next we will set the build settings for the new configuration.**
 - Inspect the toolbar to verify that the new **CheckedDebug** is the currently-active configuration.
- 1. Right-click over **Bytes** in the Solution Explorer and choose Properties. The Project Designer comes up.
- 2. Select Build from the list at the left. Verify that the Configuration is CheckedDebug. Click the Advanced button. (This button is far to the right and bottom in the dialog.) Check “Check for arithmetic overflow/underflow”.



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

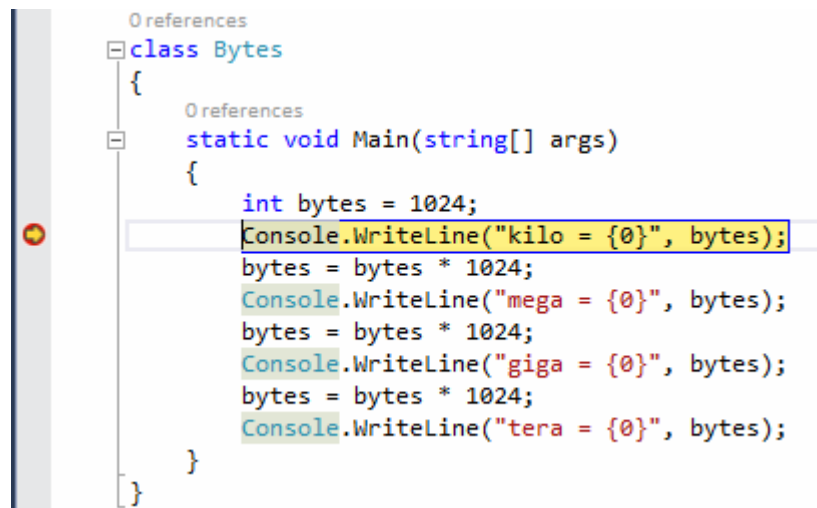
Debugging

- **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.



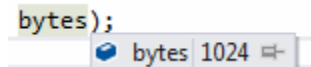
Breakpoints

- **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.

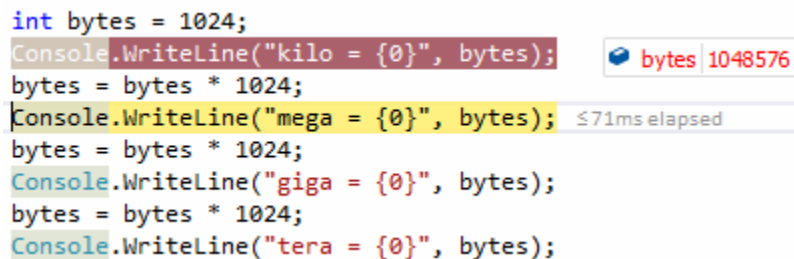


Watch Variables

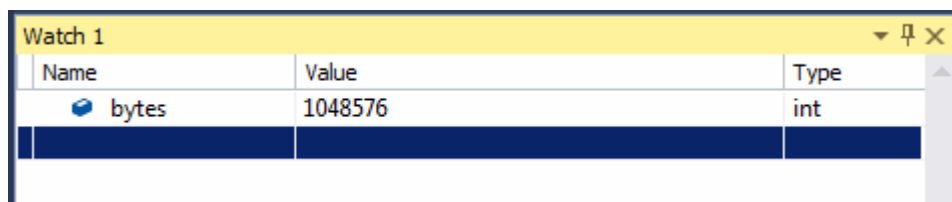
- The easiest way to inspect the value of a variable is to slide the mouse over the variable you are interested in and the value will be shown as a yellow tool tip.



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

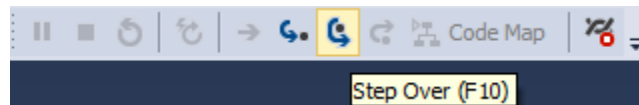


- You can also select a variable and drag it into the Watch window.
- This illustration shows a typical Watch window. You can show the Watch window through the tab at the bottom.

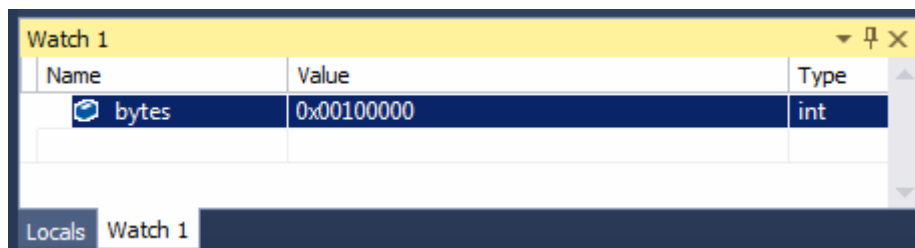


Debug Toolbar


- **The Debug toolbar gives quick access to a number of 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 can also add the Hexadecimal Display command to the Debug toolbar via Tools | Customize.
- **You will see the value in the Watch window now displayed in hex.**

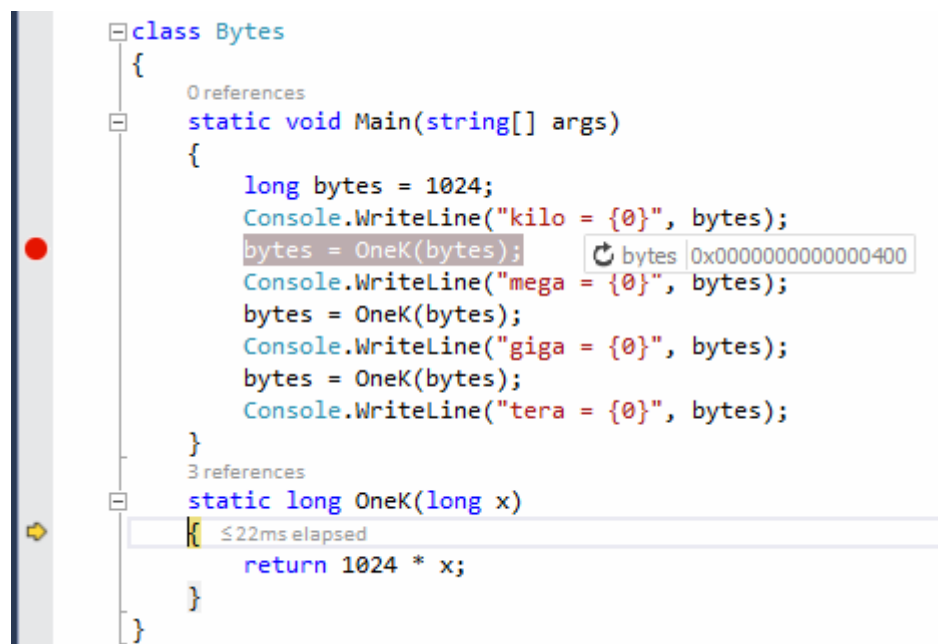


Stepping with the Debugger

- When you are stopped in the debugger, you can step through instructions either one at a time or by set amounts.
- There are a number of 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 ( Start), and then Step Into ().**
 - Note the red dot at the breakpoint and the yellow arrow in the function.

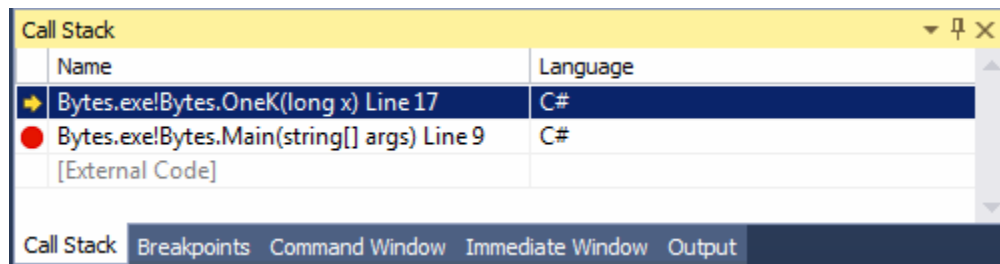


```
class Bytes
{
    0 references
    static void Main(string[] args)
    {
        long bytes = 1024;
        Console.WriteLine("kilo = {0}", bytes);
        bytes = OneK(bytes);
        Console.WriteLine("mega = {0}", bytes);
        bytes = OneK(bytes);
        Console.WriteLine("giga = {0}", bytes);
        bytes = OneK(bytes);
        Console.WriteLine("tera = {0}", bytes);
    }
    3 references
    static long OneK(long x)
    {
        ≤ 22ms elapsed
        return 1024 * x;
    }
}
```

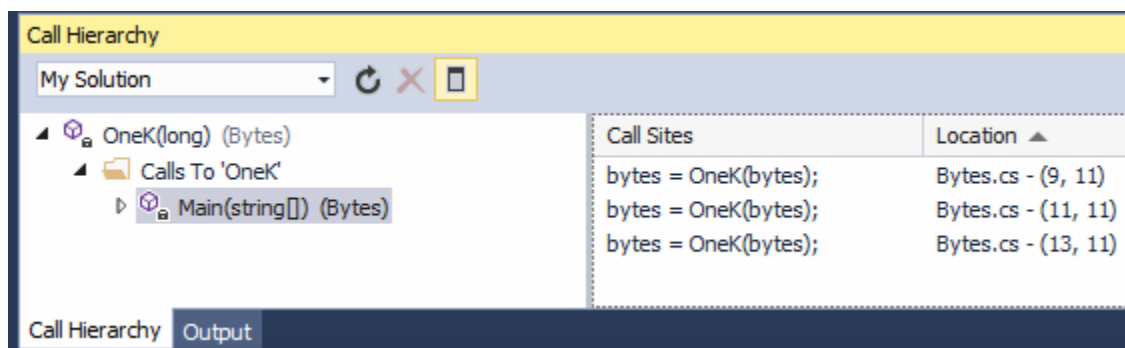
The screenshot shows the Visual Studio code editor with a C# file named `Bytes.cs`. A red dot indicates a breakpoint set on the line `bytes = OneK(bytes);` inside the `Main` method. A yellow arrow points to the `OneK` method, indicating a 'Step Into' action. A tooltip for the `bytes` variable shows its value as `0x0000000000000400` (4096). A status bar at the bottom of the `OneK` method indicates that the execution has taken `≤ 22ms` elapsed.

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, you can bring up the **Call Hierarchy**.
 - Unlike the Call Stack, which is displayed by the debugger, Call Hierarchy can be displayed at design time.



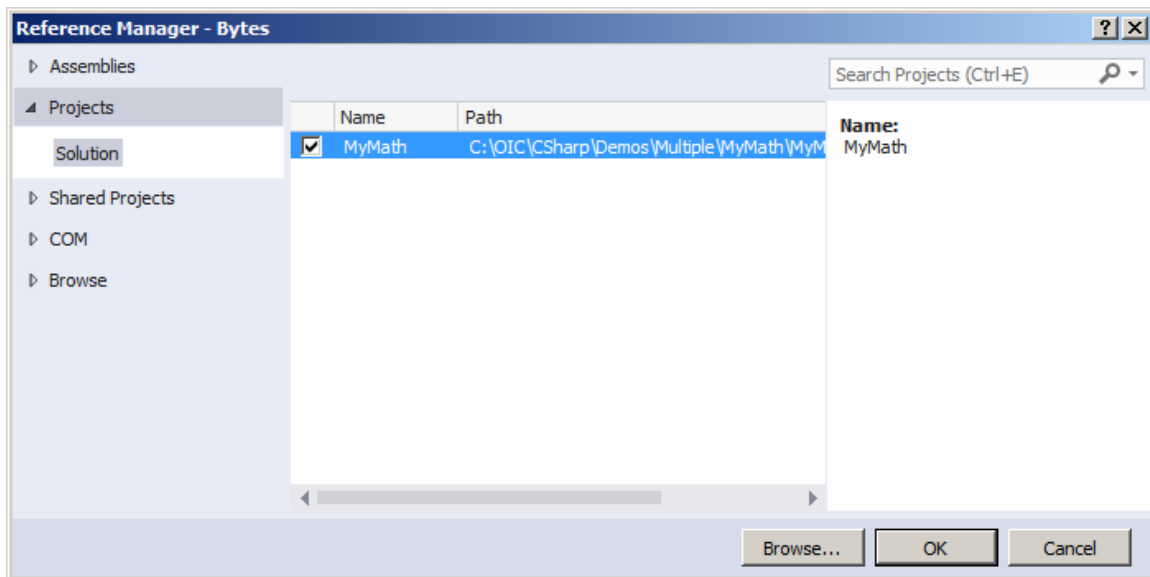
Multiple-Project Solution Demo

- **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 template and enter the name **MyMath** for the new project.
 4. In the new project, delete the file **Class1.cs**.
 5. Drag the file **SimpleMath.cs** from the **Bytes** project to the **MyMath** project. Delete the file from the **Bytes** project.
 6. Build the solution. You will get an error message.

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

Adding a Reference

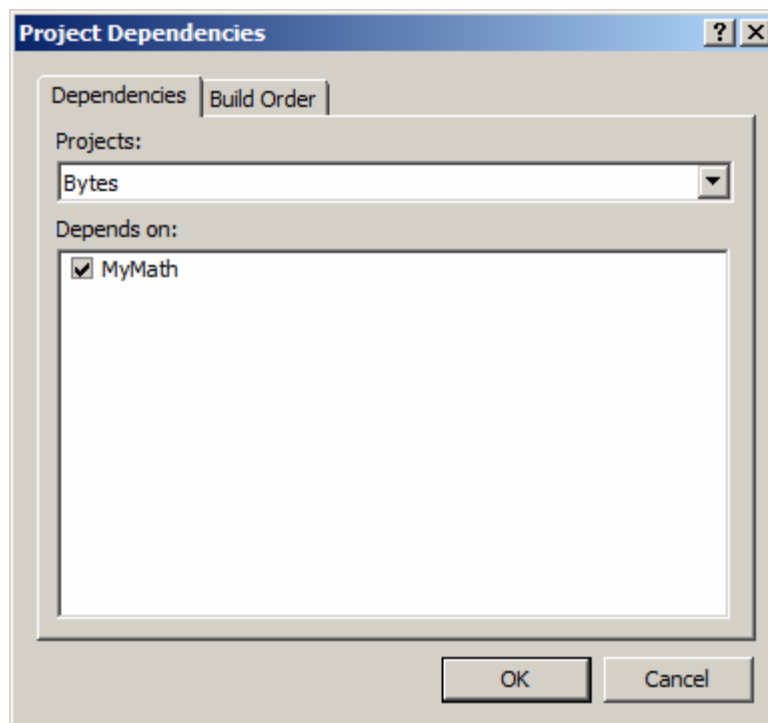
7. Right-click over References in the Bytes project and choose Add Reference from the context menu.
8. The Reference Manager dialog comes up. Select Projects under Solution. There is only one other project in this solution, **MyMath**. Select it.



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

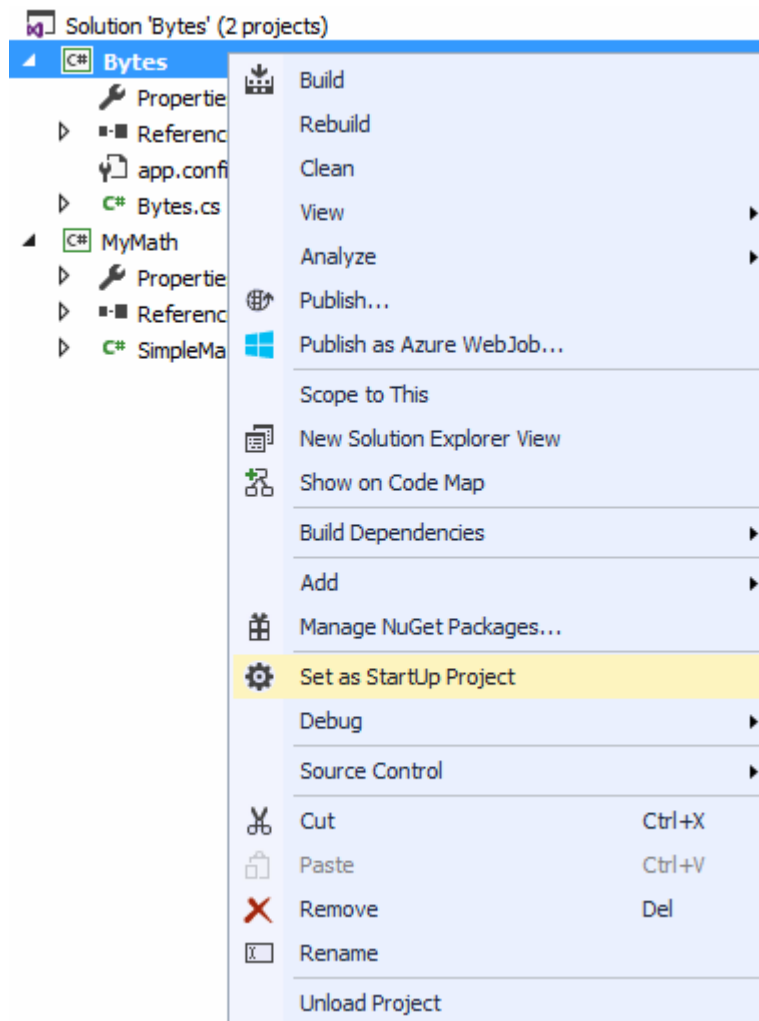
Project Dependencies

- **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

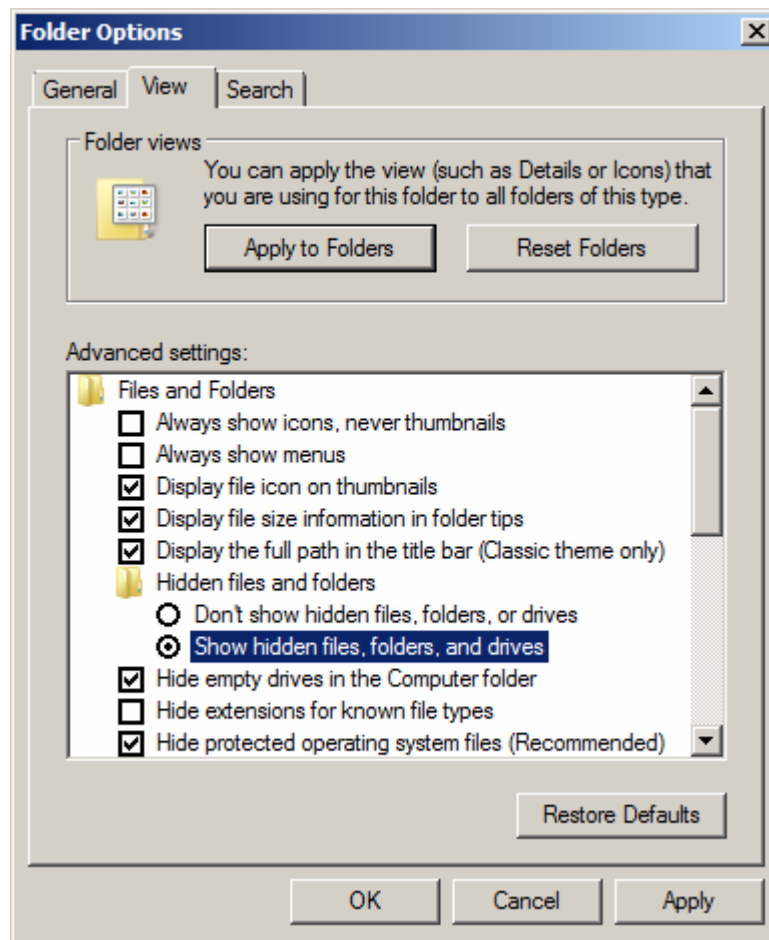
- To specify which project will run when you press F5 or Control + F5, right-click over the desired startup project and select **Set as StartUp Project** from the context menu.



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

Hidden Files

- **You can display hidden files in Windows Explorer by setting folder options appropriately.**



- **You should be aware that some Windows utilities by default will ignore hidden files.**
 - For example, WinZip will not add hidden files to a zip archive unless you explicitly ask it to.

Summary

- **Visual Studio 2015 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 2015 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.**