Chapter 3

Using Visual Studio.NET

Using Visual Studio.NET

Objectives

After completing this unit you will be able to:

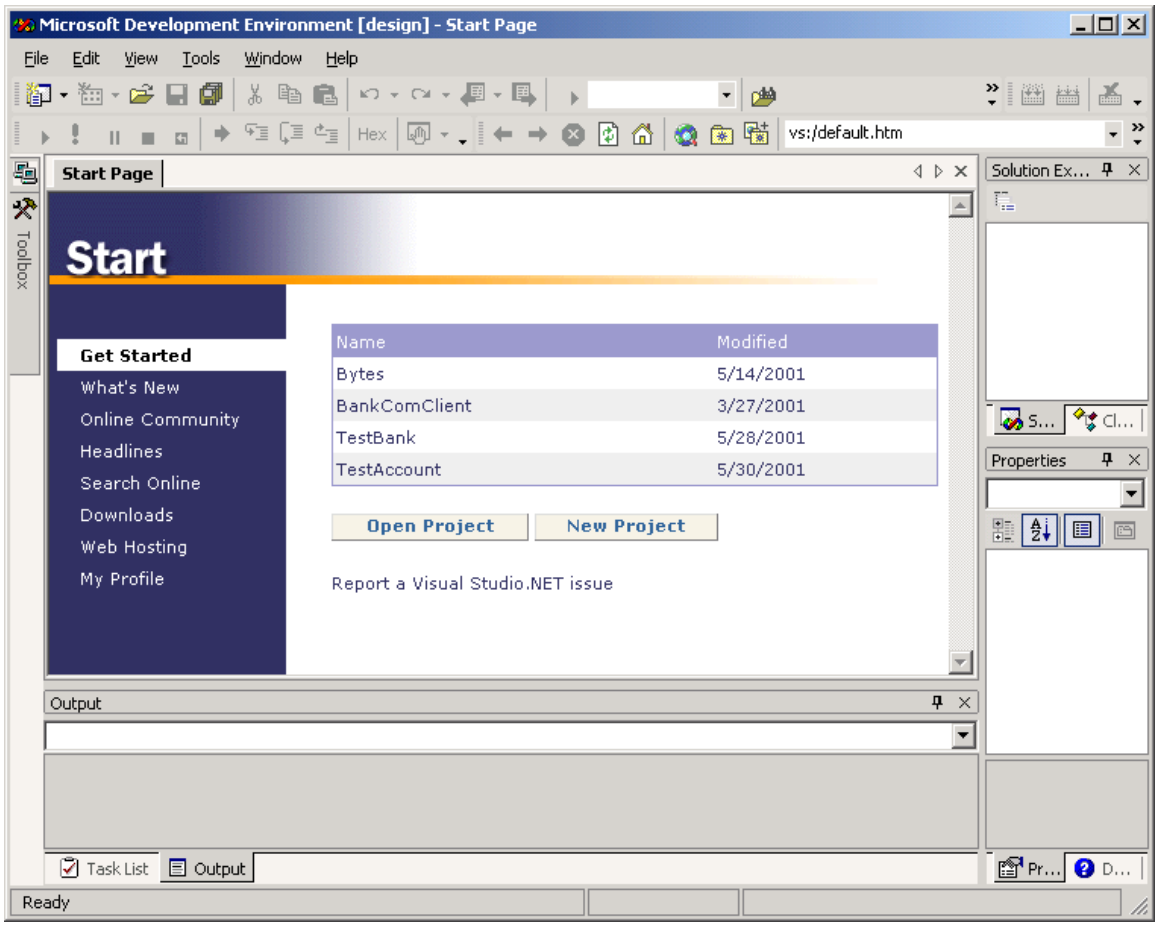
- **Use the Visual Studio.NET integrated development environment (IDE) to create, edit, debug and run C# programs.**

Visual Studio.NET

- **While it is possible to write C# programs using any text editor, and compile them with the command-line compiler, it is very tedious to program that way.**
- **An IDE makes the process of writing software much easier.**
  - An IDE provides convenience items such as a syntax-highlighting editor.
  - An IDE reduces the tedium of keeping track of configurations, environment settings, and file organizations.
- **VS.NET is a highly configurable IDE.**

Overview of VS.NET

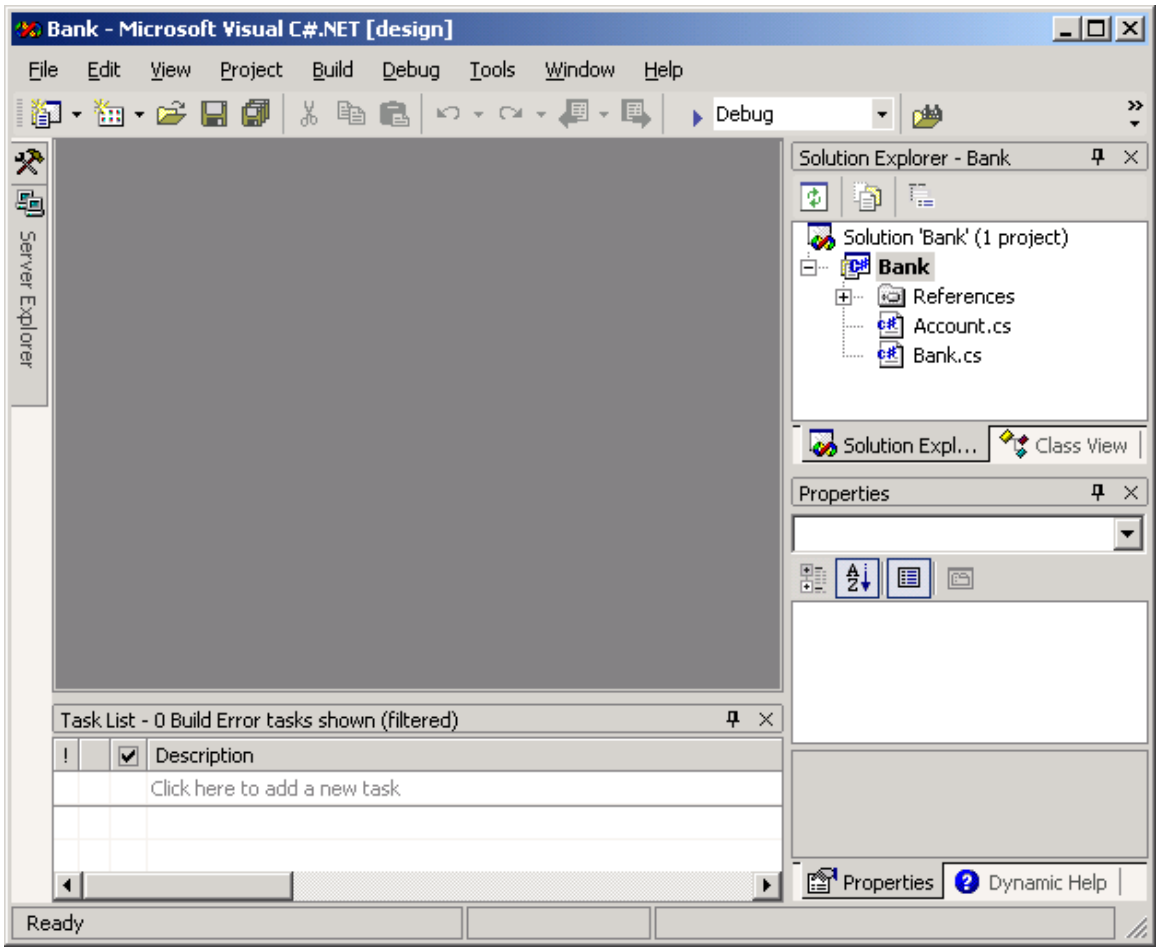
- **When you open up Visual Studio .NET, you will see a window similar to the following.**



- **The main window is an HTML page with some navigation information and configuration options.**
  - If you close this page, you can get it back from the menu: Help | Show Start Page
- **Clicking on the item "My Profile" will bring up a profile settings page.**
  - For this course, choose "Visual Studio Developer".

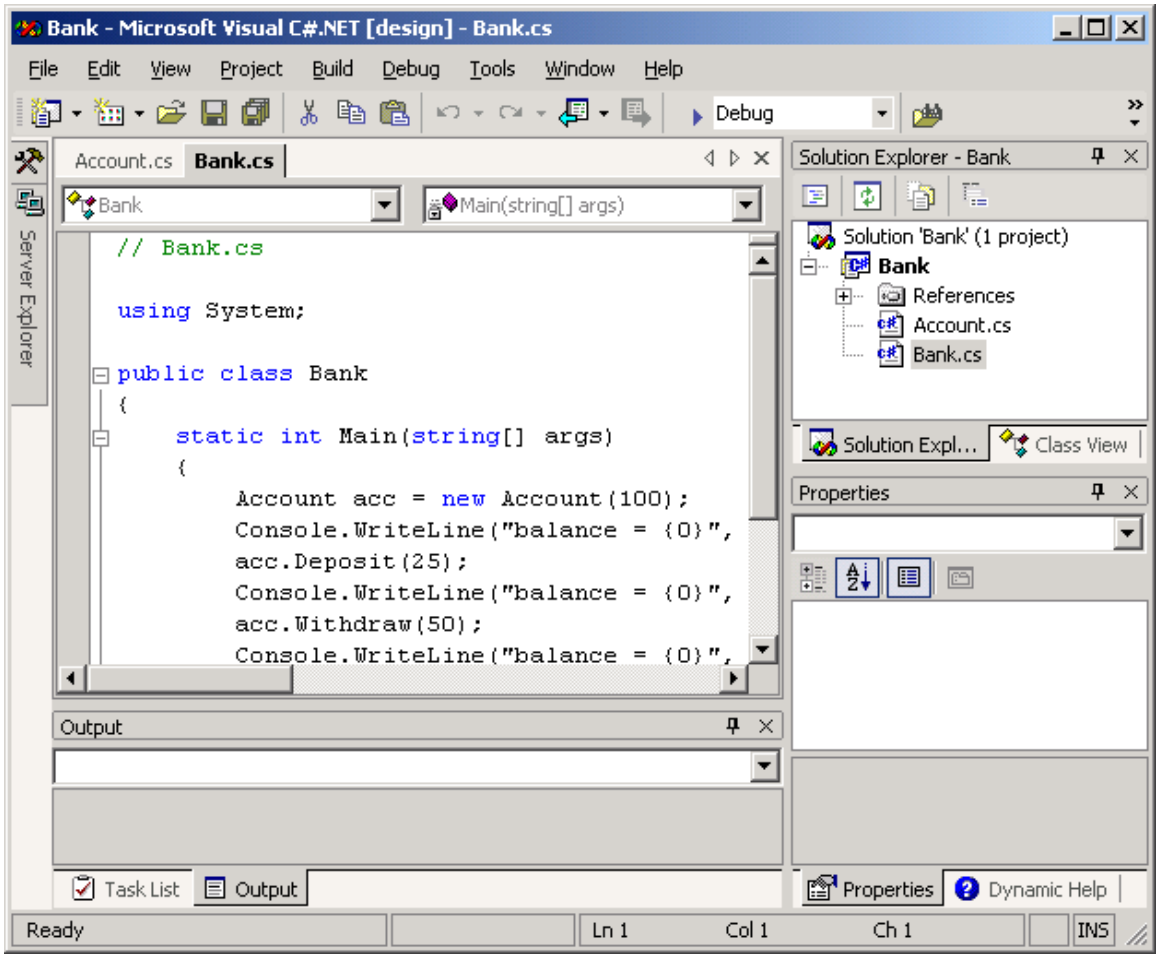
Overview of VS.NET (Cont'd)

- **This module 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
  - Navigate to directory OIC\Chap03\Bank
  - Select the file **Bank.sln**, and open it.



Overview of VS.NET (Cont'd)

- **On the top right is the Solution Explorer, which shows you the structure of your *solution* (which may consist of several *projects*).**
- **Double-click on each of *Account.cs* and *Bank.cs*, the two source files in the Bank project.**



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 to specify which toolbars are shown, choose View | Toolbar, 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

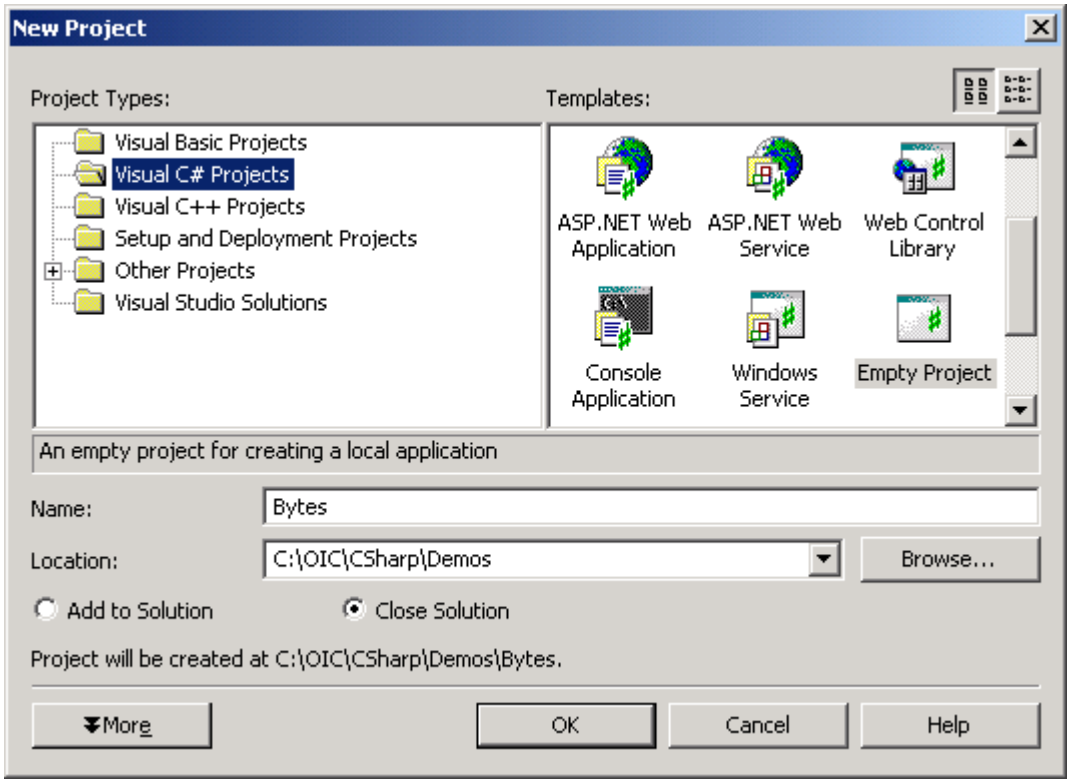
- **To illustrate how to customize a toolbar, we will add the "Start Without Debugging" command (a red exclamation point) to the Debug toolbar.**

1. Select menu Tools | Customize... to bring up the Customize dialog.
2. Select the Commands tab.
3. In Categories, select Debug, and in Commands select Start Without Debugging.
4. Drag the selected command onto the Debug toolbar, positioning it to the immediate right of the wedge shaped Start  button.
5. Close the Customize dialog.

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. From Visual Studio main menu choose File | New | Project.... This will bring up the New Project dialog.
2. For Project Types choose "Visual C# Projects" and for Templates choose "Empty Project."
3. Click the Browse button, navigate to **Demos**, and click Open.
4. In the Name field, type **Bytes**. See below. Click OK.



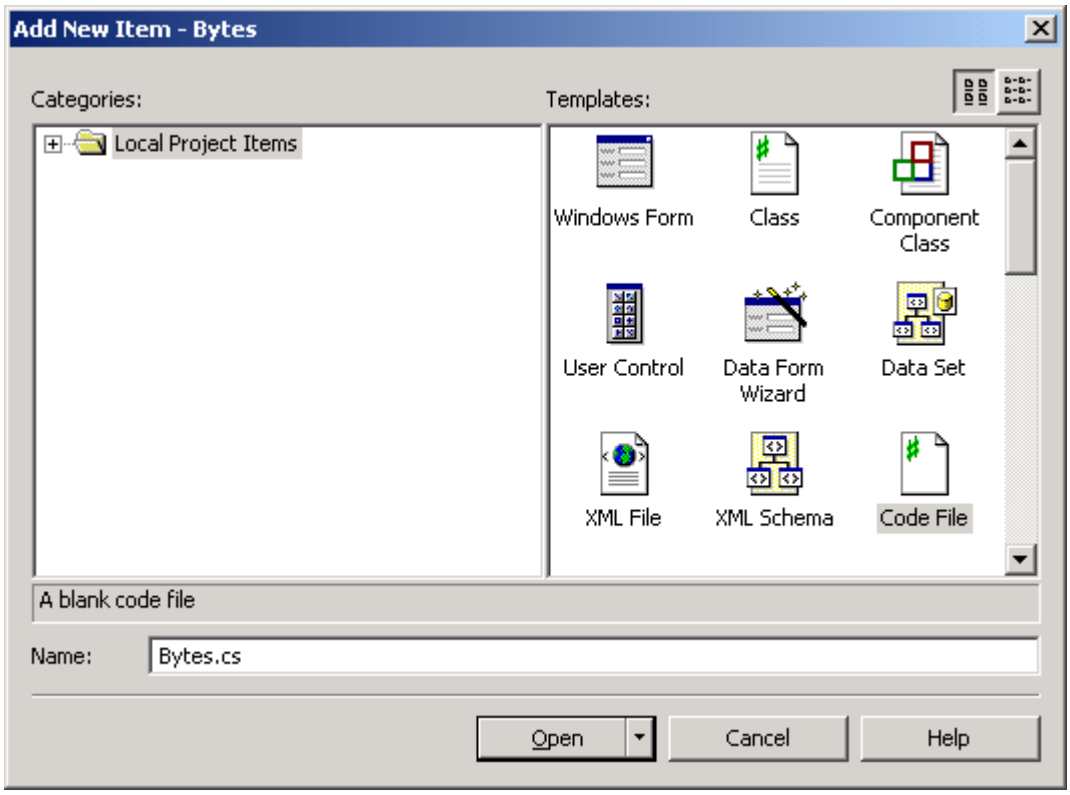
Adding a C# File

- **At this point you will have an empty C# project. 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 | Add New Item.... This will bring up the Add New Item dialog.

2. For Categories choose "Local Project Items" and for Templates choose "Code File."

3. For Name type **Bytes.cs**. Click Open.



Using the Visual Studio Text Editor

- **In the Solution Explorer, double-click on *Bytes.cs*.**
- **Enter the following program into the empty file in the Visual Studio text editor.**

```
// Bytes.cs

using System;

public class Bytes
{
   public static int 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);
      return 0;
   }
}
```

  - Notice that the Visual Studio text editor highlights syntax, indents automatically, and even inserts matching closing braces for you!

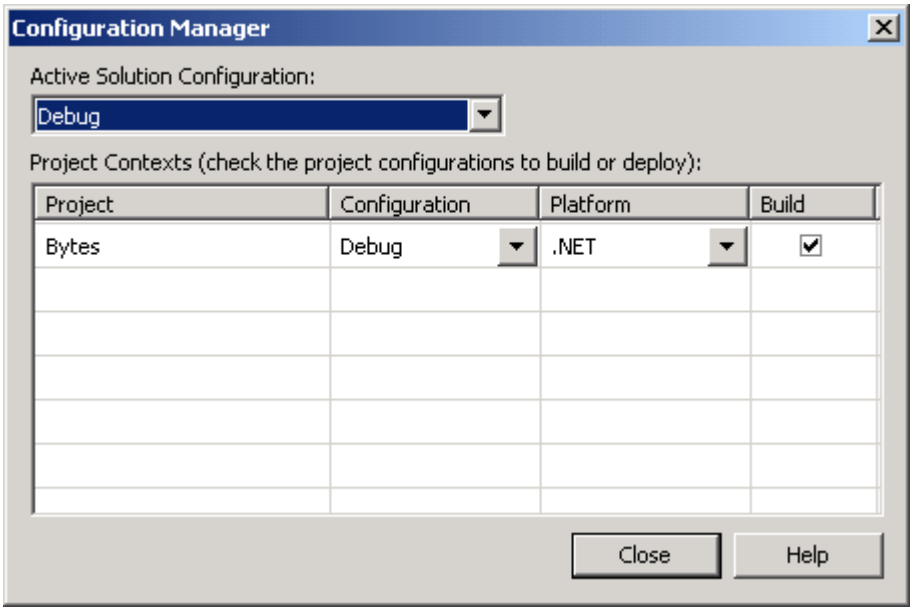
Build and Run the Bytes Project

- **You can build the project by using one of the following:**
  - Menu Build | Build
  - Toolbar 
  - Keyboard shortcut Ctrl + Shift + B
- **You can run the program by using one of the following:**
  - Menu Debug | Start Without Debugging
  - Toolbar
  - Keyboard shortcut Ctrl + F5
- **You can run the program in the debugger by using one of the following:**
  - Menu Debug | Start
  - Toolbar 
  - Keyboard shortcut F5
- **Try it!**

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** 

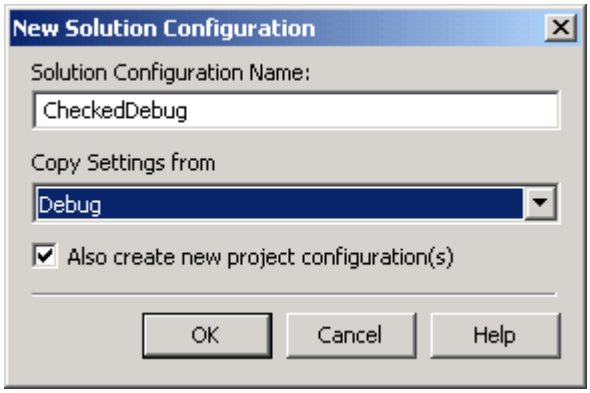
or using the menu Build | Configuration Manager..., which will bring up the Configuration Manager dialog (shown below).



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 (we will discuss "checked" builds in a later module).**

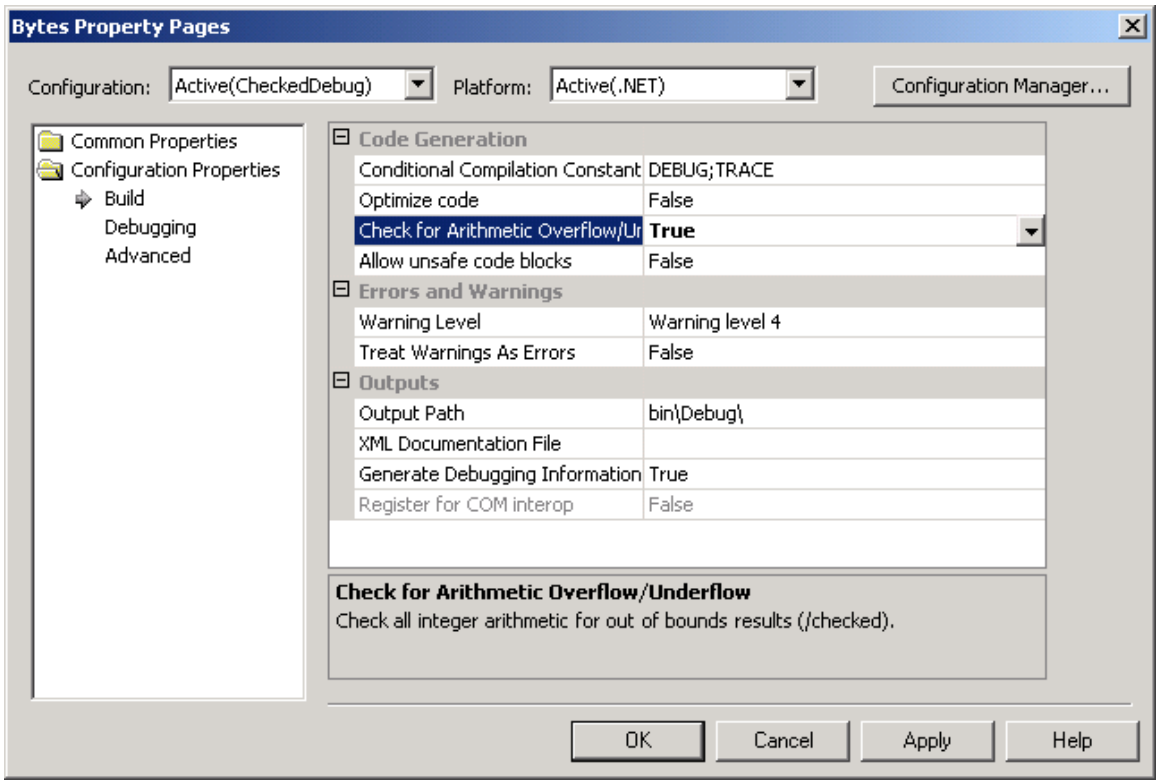
1. Bring up the Configuration Manager dialog.
2. From the Active Solution Configuration: dropdown, choose **<New...>**. The New Solution Configuration dialog will come up.
3. Type **CheckedDebug** as the configuration name. Choose Copy Settings from **Debug**. Check "Also create new project configuration(s)" (see below). Click OK.



Setting Build Settings for a Configuration

- **Next we will set the build settings for the new configuration.**
  - Check 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 "Bytes Property Pages" dialog comes up.
2. In Configuration Properties, select Build. Change the setting for "Check for overflow underflow" to **True** (see below). Click OK.

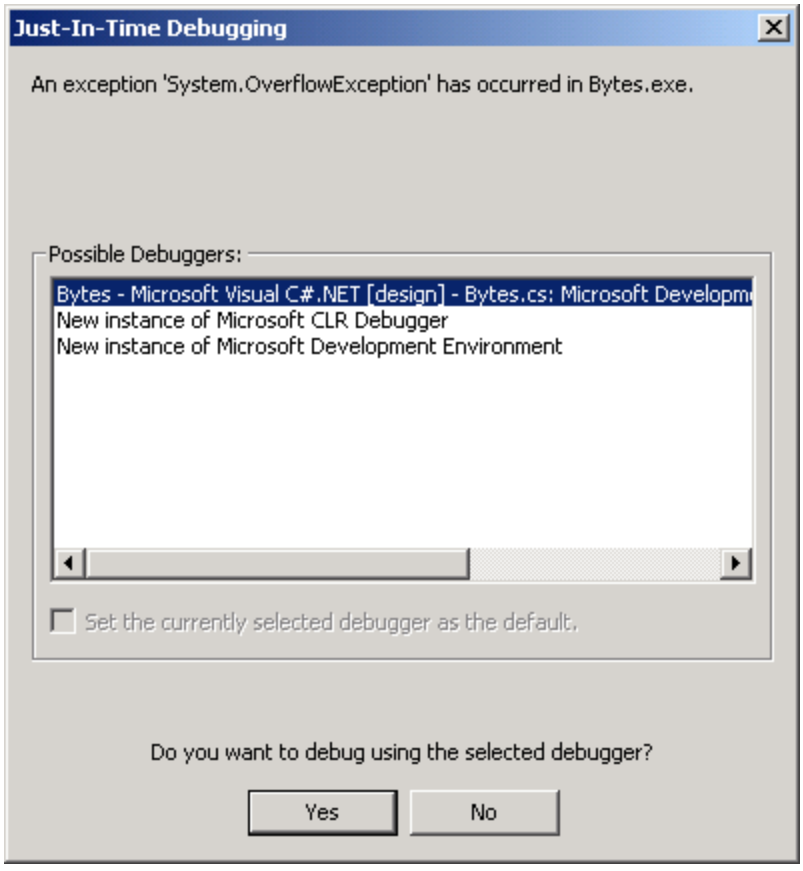


Debugging

- **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:**
  - 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.
  - Standard Debugging. You start the program under the debugger. You may set breakpoints, single step, and so on.

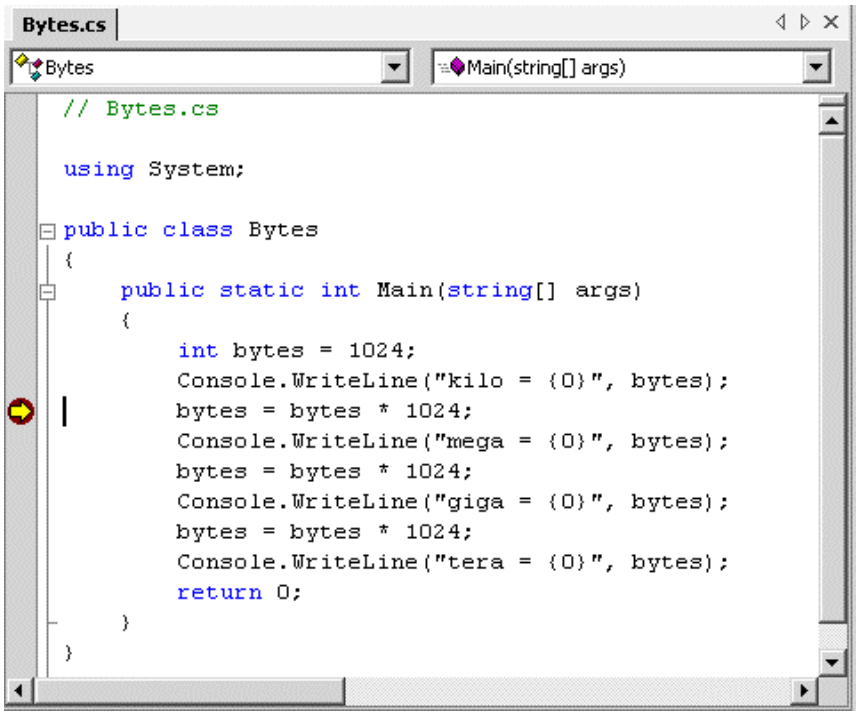
Just-in-Time Debugging

- **Build and run (without debugging) the *Bytes* program from the previous section, making sure to use the *CheckedDebug* configuration.**
- **This time the program will not run through smoothly to completion, but an exception will be thrown.**
  - A "Just-In-Time Debugging" dialog will be shown (see below). Click Yes to debug.



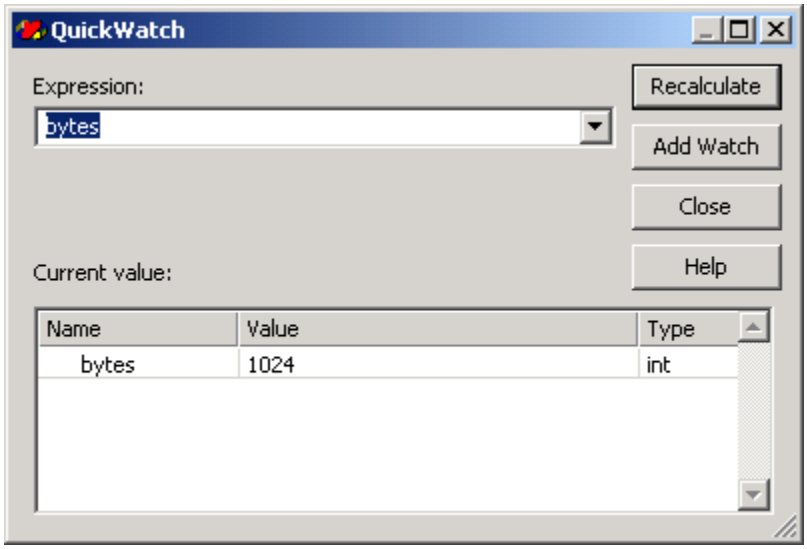
Standard Debugging -- Breakpoints

- **The way you typically do 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 can also set the cursor on the desired line and click the "hand" toolbar button  to toggle a breakpoint (set if not set, and remove if a breakpoint is set).
  - If you want to remove all breakpoints, you can use the menu Debug | Clear All Breakpoints, or you can use the toolbar button 
  - A yellow arrow over the red dot of the breakpoint shows where the breakpoint has been hit.

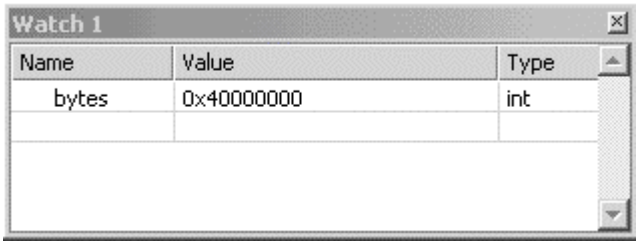


Standard Debugging -- Watch Variables

- **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.**
- **You can also right-click over a variable and choose Quick Watch (or use the eyeglass toolbar button).**
  - The illustration shows a typical Quick Watch window.

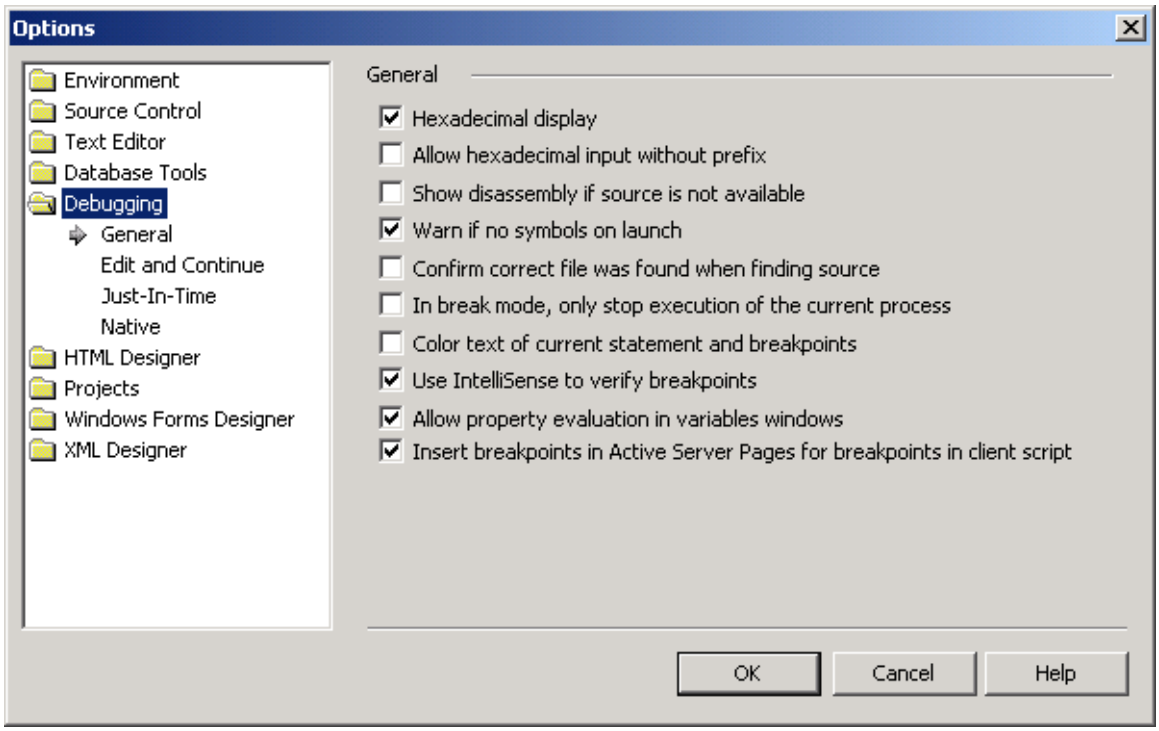


- **When you are stopped in the debugger, you can add a variable to the Watch window by right-clicking over it and choosing Add Watch.**



Debugger Options

- **You can change debugger options from the menu Tools | Options, and select Debugging from the list.**

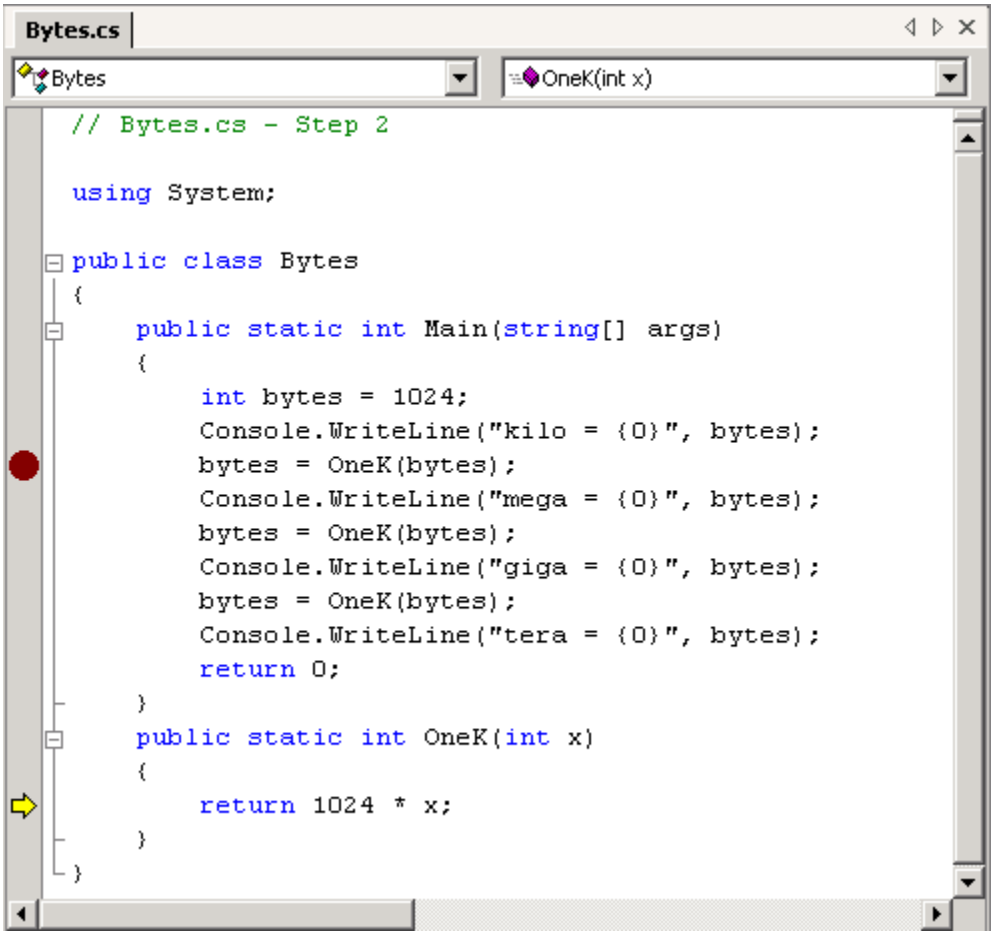


Stepping with the Debugger

- **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 most common are (in the order shown on the toolbar):**
  - Step Into
  - Step Over
  - Step Out
- **There is also a Run to Cursor button**  **.**

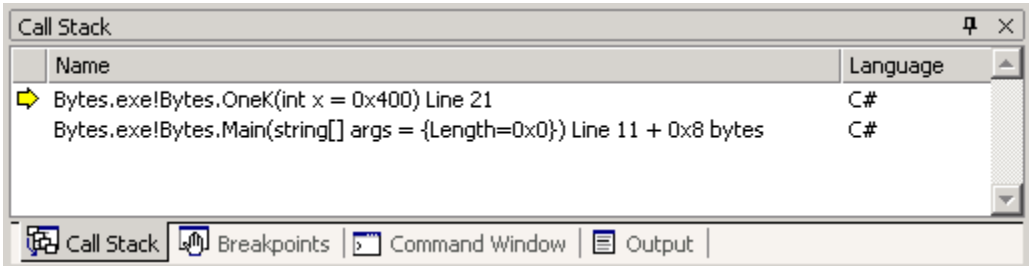
Demo: Stepping with the Debugger

- **Build the Bytes\Step2 project.**
  - The multiplication by 1024 has been replaced by a function.
- **Set a breakpoint at the first function call, start the program ( ), and then Step Into ( ).**
  - Note the red dot at the breakpoint and the yellow arrow in the function.



The Call Stack

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



Summary

- **Visual Studio.NET is a very rich integrated development environment (IDE), with many features to make programming more enjoyable.**
- **In this module 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.**