T H R E E

Visual Studio.NET

Although it is possible to program .NET using only the command line compiler, it is much easier and more enjoyable to use Visual Studio.NET. In this chapter we cover the basics of using Visual Studio to edit, compile, run, and debug programs. You will then be equipped to use Visual Studio in the rest of the book. This chapter covers the basics to get you up and running using Visual Studio. We will introduce additional features of Visual Studio later in the book as we encounter a need. This book was developed using beta software, and in the final released product you may encounter some changes to the information presented here. Also, Visual Studio is a very elaborate Windows application that is highly configurable, and you may encounter variations in the exact layout of windows, what is shown by default, and so on. As you work with Visual Studio, a good attitude is to see yourself as an explorer discovering a rich and varied new country.

OVERVIEW OF VISUAL STUDIO.NET

Open up Microsoft Visual Studio.NET 7.0 and you will see a starting window similar to what is shown in Figure 3–1.

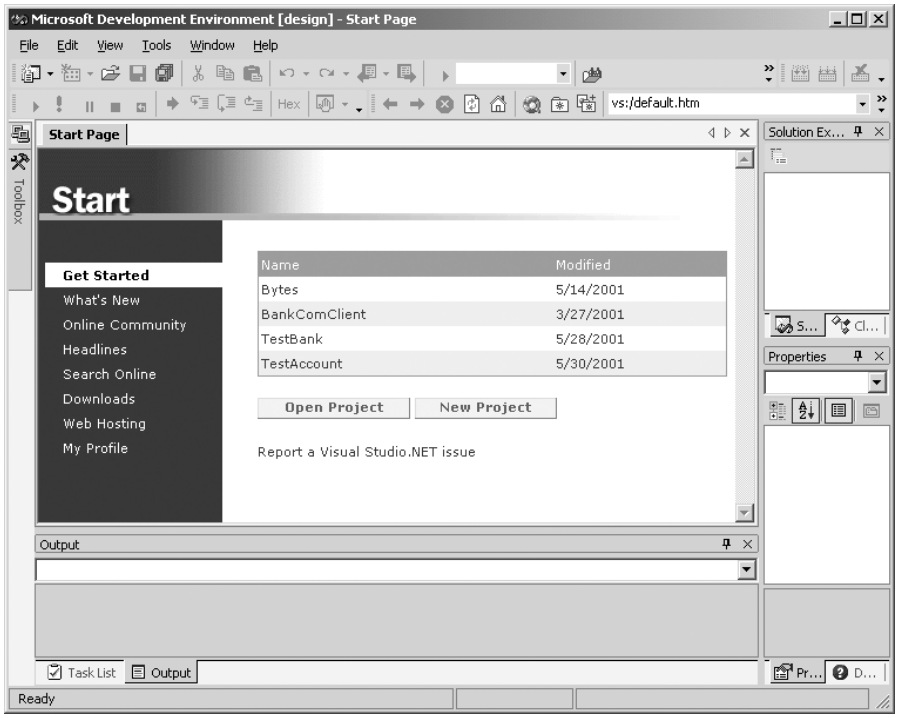


Figure 3–1 *Visual Studio.NET main window.*

What you see on default startup is the main window with an HTML page that can help you navigate among various resources, open or create projects, and change your profile information. (If you close the start page, you can get it back anytime from the menu Help | Show Start Page.) Clicking on **My Profile** will bring up a profile page on which you can change various settings. There is a standard profile for "typical" work in Visual Studio ("Visual Studio Developer" profile), and special ones for various languages. Since Visual Studio.NET is the unification of many development environments, programmers used to one particular previous environment may prefer a particular keyboard scheme, window layout, and so on. For example, if you choose the profile "Visual Basic Developer," you will get the Visual Basic 6 keyboard scheme. In this book we will use all the defaults, so go back to the profile "Visual Studio Developer" if you made any changes. See Figure 3–2.

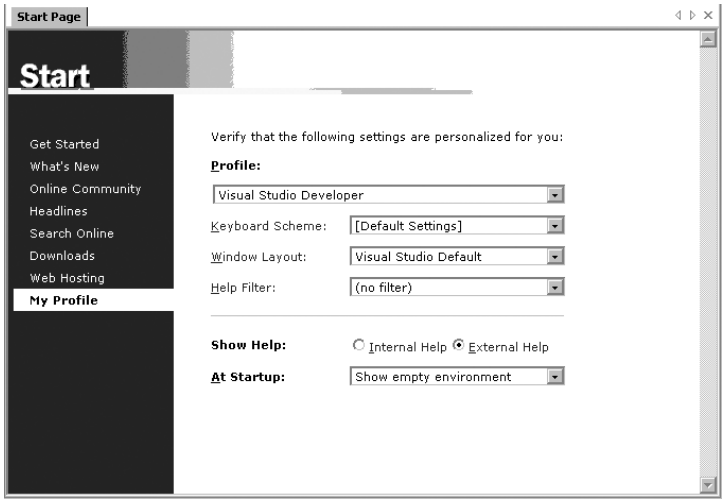


Figure 3–2 *Visual Studio.NET profile page.*

To gain an appreciation of some of the diverse features in Visual Studio.NET, open up the **Bank** console solution in this chapter (File | Open Solution..., navigate to the **Bank** directory, and open the file **Bank.sln**). You will see quite an elaborate set of windows. See Figure 3–3.

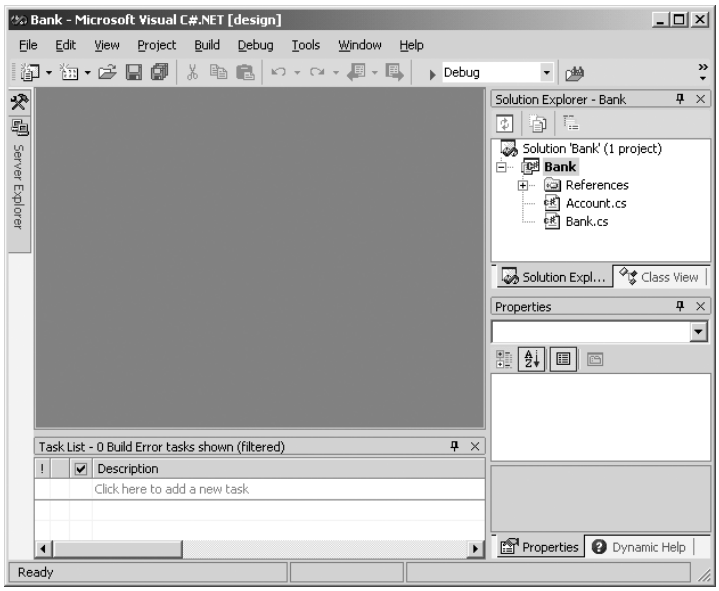


Figure 3–3 *A console project in Visual Studio.NET.*

Starting from the left are icons for the Server Explorer and the Toolbox, followed by the main window area, which currently is just a gray area. Underneath the main window is the Output Window, which shows the results of builds and so on. Continuing our tour, on the top right is the Solution Explorer, which enables you to conveniently see all the files in a "solution," which may consist of several "projects." On the bottom right is the Properties window, which lets you conveniently edit properties on forms for Windows applications. The Properties window is very similar to the Properties Window in Visual Basic.

From the Solution Explorer you can navigate to files in the projects. In turn, double-click on each of **Account.cs** and **Bank.cs,** the two source files in the **Bank** project. Text editor windows will be brought up in the main window area. Across the top of the main window are horizontal tabs to quickly select any of the open windows. Visual Studio.NET is a Multiple Document Interface (MDI) application, and you can also select the window to show from the Windows menu. Figure 3–4 shows the open source files with the horizontal tabs.

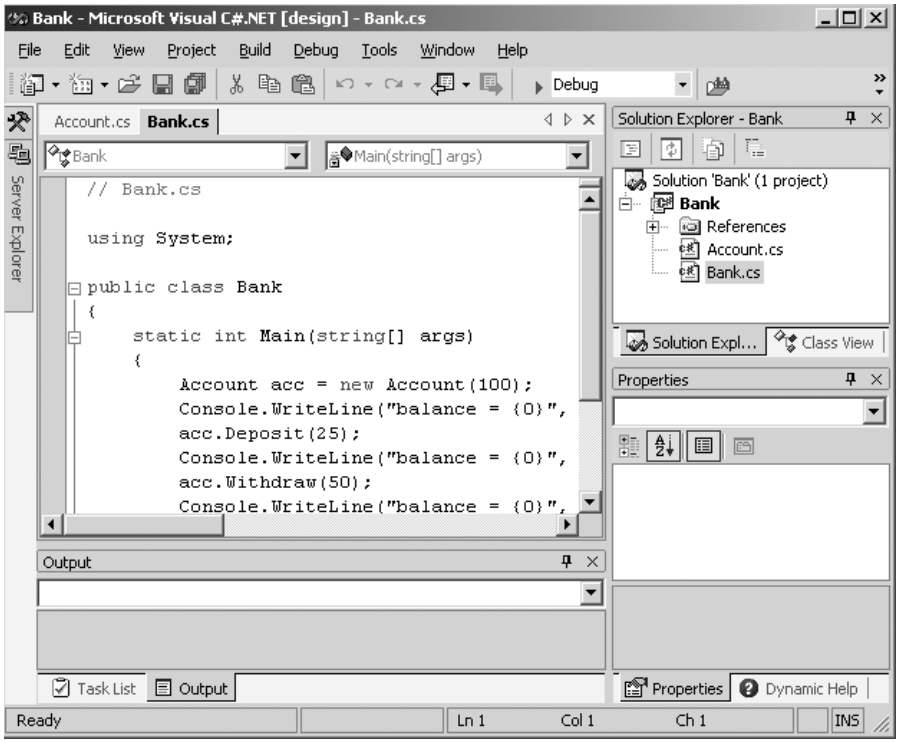


Figure 3–4 *Horizontal tabs for open source files.*

Toolbars

Visual Studio comes with many different toolbars. You can configure which toolbars you wish displayed, and you can drag toolbars to position them to where you find them most convenient. You can also customize toolbars by adding or deleting buttons that correspond to different commands.

To specify which toolbars are displayed, bring up the menu View | Toolbars. You can also right click in any empty area of a toolbar. There will be a check mark next to the toolbars which are currently displayed. By clicking on an item on this menu you can make the corresponding toolbar button appear or disappear. For your work in this book add the toolbars,

- Build
- Debug

CUSTOMIZING A TOOLBAR

We want to make sure that the "Start Without Debugging" command is available on the Debug toolbar. If it is not already on your Debug toolbar (it is a red exclamation point), you can add it by the following procedure, which can be used to add other commands to toolbars.

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. See Figure 3-5.
4. Drag the selected command onto the Debug toolbar, positioning it where you desire. Place it to the immediate right of the wedge-shaped Start button.
5. Close the Customize dialog.

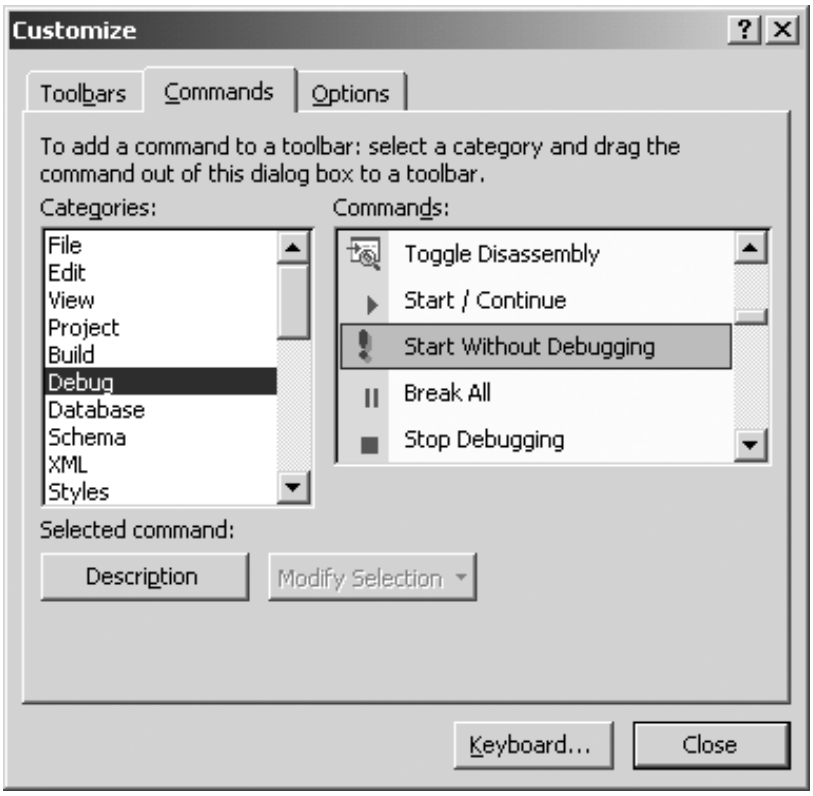


Figure 3–5 *Adding a new command to a toolbar.*

CREATING A CONSOLE APPLICATION

As our first exercise in using Visual Studio, we will 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. If you want to follow along on your PC as you read, you can use the **Demos** directory for this chapter. The first version is in **Bytes\Step1**. A final version can be found in **Bytes\Step3**.

Creating a C# Project

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 Figure 3–6. Click OK.

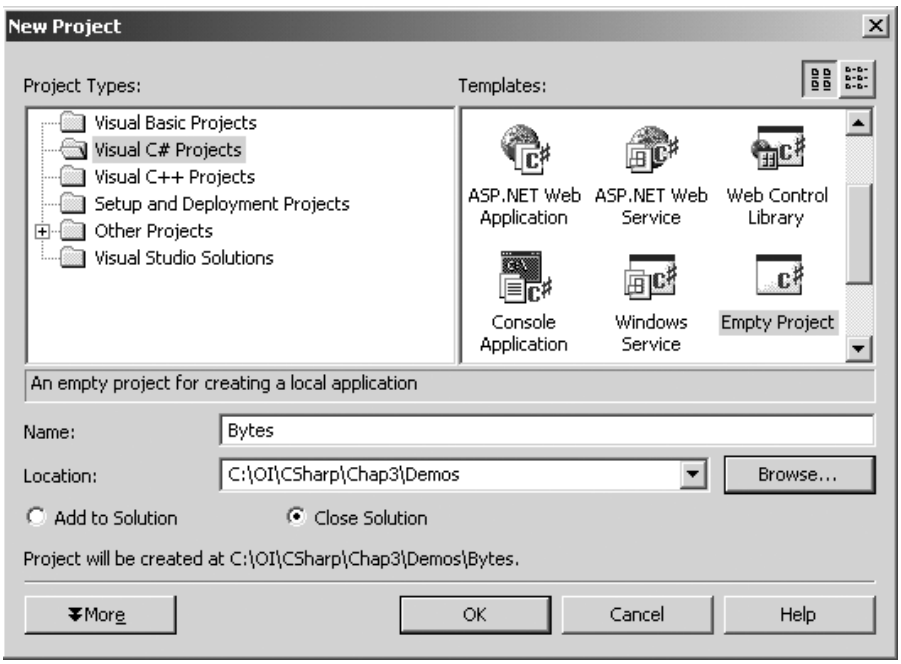


Figure 3–6 *Creating an empty C# project.*

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.** See Figure 3–7. Click Open.

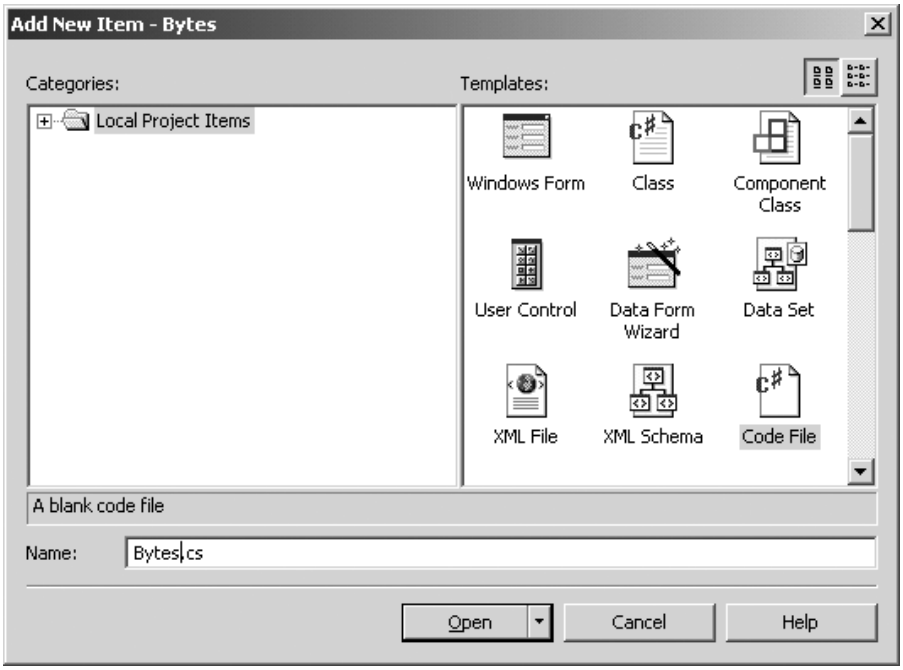


Figure 3–7 *Adding an empty C# file to a C# project.*

Using the Visual Studio Text Editor

In the Solution Explorer double-click on **Bytes.cs**. This will open up the empty file **Bytes.cs** in the Visual Studio text editor. Type in the following program, and notice things like color syntax highlighting as you type.

```
// 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;
  }
}
```

Besides the color syntax highlighting, other features include automatic indenting and putting in a closing right curly brace to match the left curly brace you type. All in all, you should find the Visual Studio editor friendly and easy to use.

Building the Project

You can build the project by using one of the following:

- Menu Build | Build
- Toolbar 
- Keyboard shortcut Ctrl + Shift + B

Running the Program

You can run the program by using one of the following:

- Menu Debug | Start Without Debugging
- Toolbar
- Keyboard shortcut Ctrl + F5

You will see the following output in a console window that opens up:

```
kilo = 1024
mega = 1048576
giga = 1073741824
tera = 0
Press any key to continue
```

We will investigate the reason for the strange output later. If you press any key, as indicated, the console window will close.

Running the Program in the Debugger

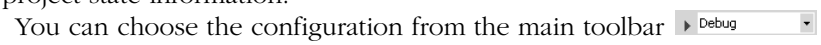
You can run the program in the debugger by using one of the following:

- Menu Debug | Start
- Toolbar 
- Keyboard shortcut F5

A console window will briefly open up and then immediately close. If you want the window to stay open, you must explicitly program for it, for example, by asking for input. You can set a breakpoint to stop execution before the program exits. We will outline features of the debugger later in the chapter.

PROJECT CONFIGURATIONS

A project *configuration* specifies build settings for a project. You can have several different configurations, and each configuration will be built in its own directory, so you can exercise the different configurations independently. Every project in a Visual Studio solution has two default configurations, **Debug** and **Release**. As the names suggest, the **Debug** configuration will build a debug version of the project, where you can do source level debugging by setting breakpoints, and so on. The **bin\Debug** directory will then contain a *program database* file with a **.pdb** extension that holds debugging and project state information.

You can choose the configuration from the main toolbar 

. You can also choose the configuration using the menu Build | Configuration Manager..., which will bring up the Configuration Manager dialog. From the Active Solution Configuration dropdown, choose **Release.** See Figure 3–8.

Build the project again. Now a second version of the IL language file **Bytes.exe** is created, this time in the **bin\Release** directory. There will be no **.pdb** file in this directory.

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. As we will discuss in Chapter 5, if you build with the **/checked** compiler switch, the compiler will generate IL code to check for integer underflow and overflow. In Visual Studio you set compiler options through dialog boxes. The following steps will guide you through creating a new configuration called **CheckedDebug** that will build a checked version of the program.

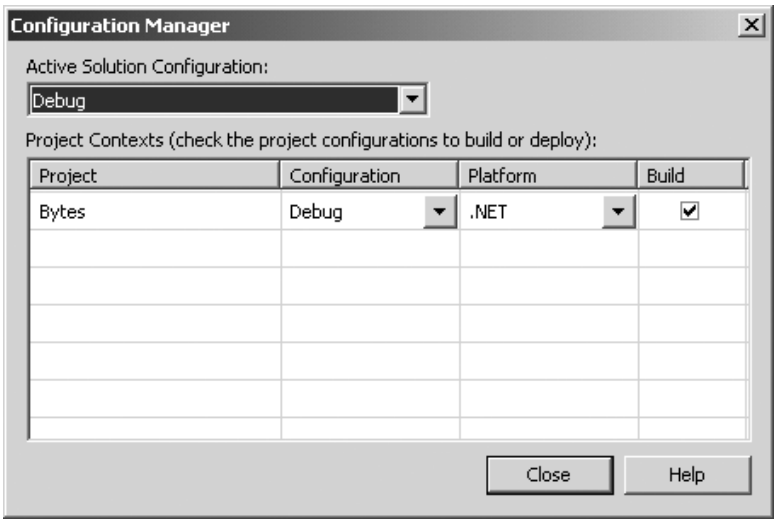


Figure 3–8 *Choosing Release in the Configuration Manager.*

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 Figure 3–9. Click OK.

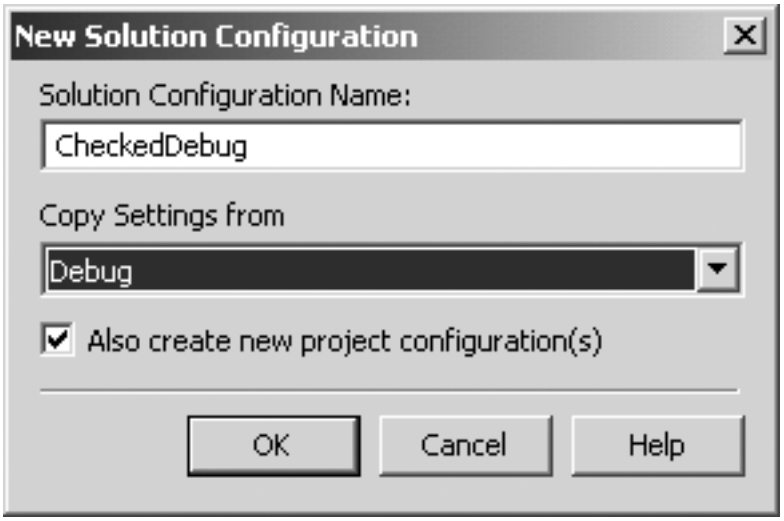


Figure 3–9 *Creating a new configuration.*

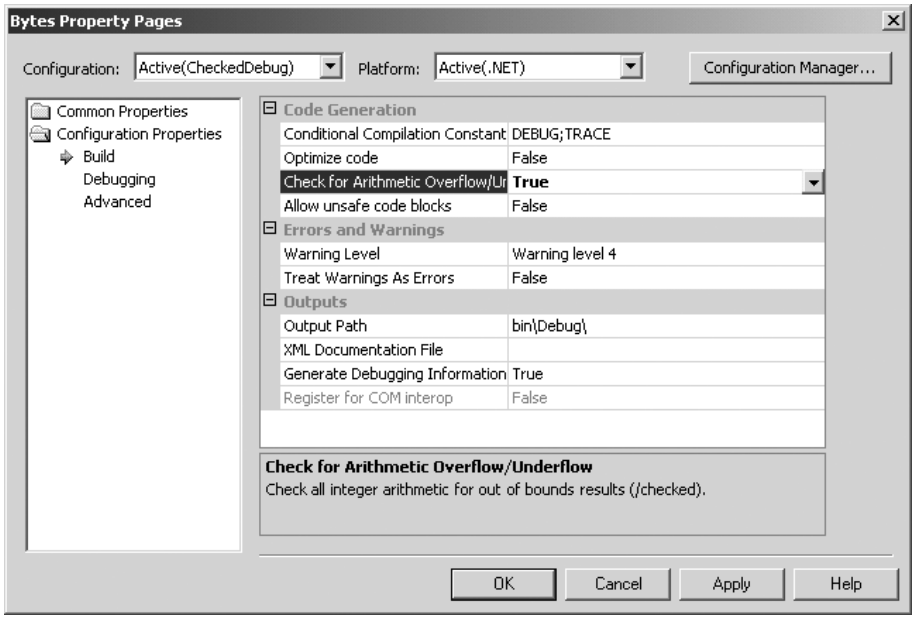


Figure 3–10 *Changing the build settings for a configuration.*

Setting Build Settings for a Configuration

Next we will set the build settings for the new configuration. (You could also set build settings for one of the standard configurations, if you wanted to make any changes from the defaults provided.) 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 Figure 3–10). Click OK.

DEBUGGING

In this section we will discuss some of the debugging facilities in Visual Studio. 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 Figure 3–11). Click Yes to debug.

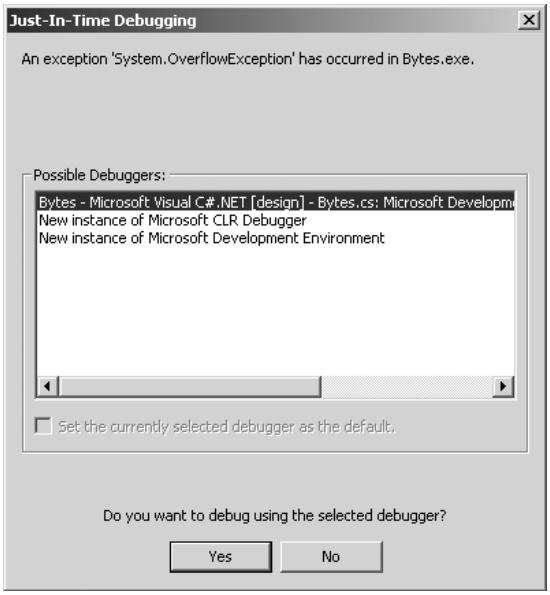


Figure 3–11 *Just-In-Time Debugging dialog is displayed in response to an exception.*

Click OK in the "Attach to Process" dialog and then click Break in the "Microsoft Development Environment" dialog. You will now be brought into a window showing the source code where the problem arose, with an arrow pinpointing the location.

To stop debugging you can use the ■ toolbar button or the menu Debug | Stop Debugging.

Standard Debugging

BREAKPOINTS

The way you typically do standard debugging is to set a breakpoint and then run using the debugger. As an example, set a breakpoint at the first line:

```
bytes = bytes * 1024;
```

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). Now you can run under the debugger, and the breakpoint should be hit. A yellow arrow over the red dot of the breakpoint shows where the breakpoint has been hit. See Figure 3–12.

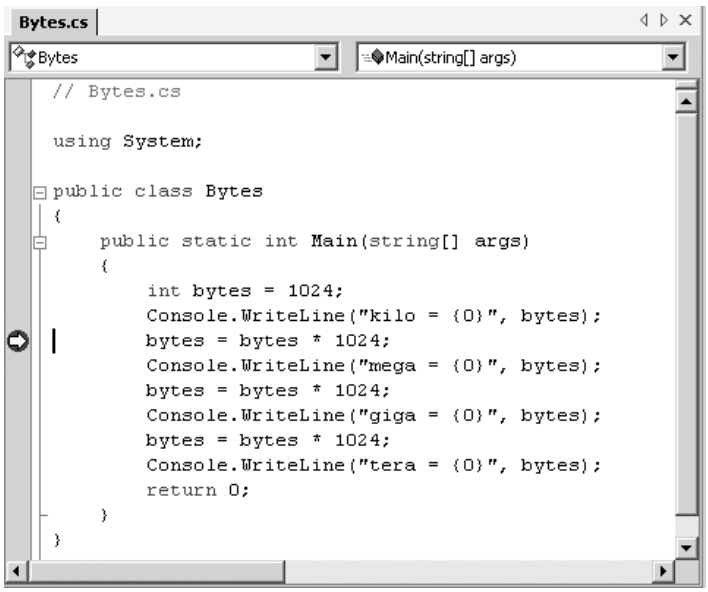


Figure 3–12 *A breakpoint has been hit.*

When you are done with a breakpoint, you can remove it by clicking again in the gray bar or by toggling with the hand toolbar button. If you want to remove all breakpoints, you can use the menu Debug | Clear All Breakpoints, or you can use the toolbar button .

WATCHING VARIABLES

At this point you can inspect variables. The easiest way 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 eyeglasses toolbar button). Figure 3–13 shows a typical Quick Watch window. You can also change the value of a variable from this 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. The Watch window can show a number of variables, and the Watch window stays open as the program executes. When a variable changes value, the new value is

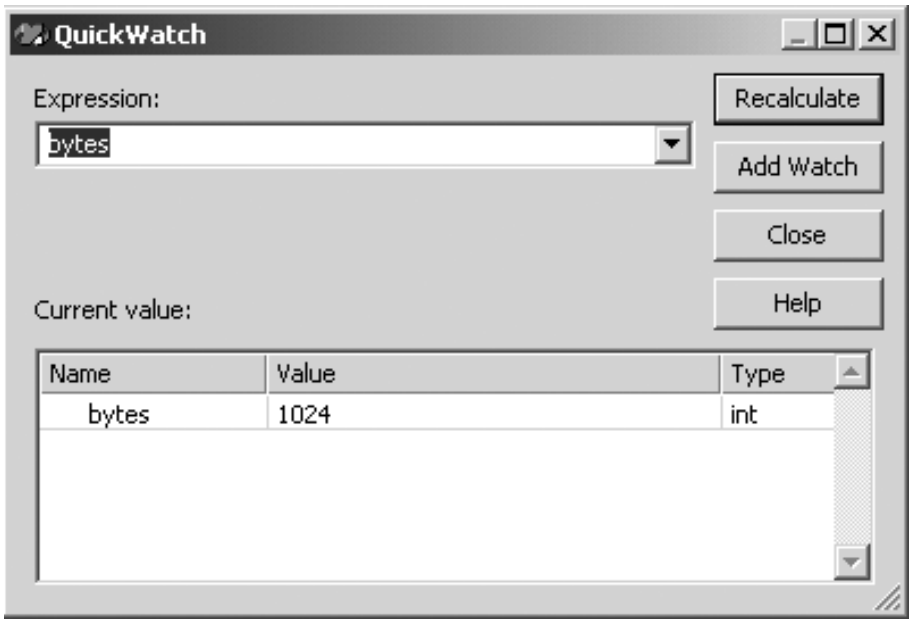


Figure 3–13 *Quick Watch window shows variable, and you can change it.*

shown in red. Figure 3–14 shows the Watch window (note that the display has been changed to hex, as described in the next section).

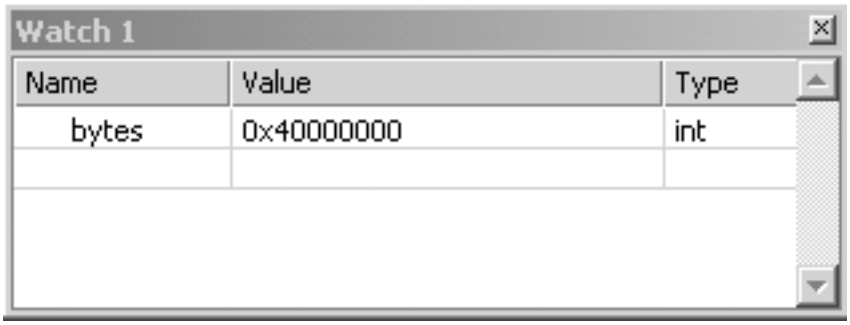


Figure 3–14 *Visual Studio Watch window.*

DEBUGGER OPTIONS

You can change debugger options from the menu Tools | Options, and select Debugging from the list. Figure 3–15 illustrates setting a hexadecimal display. If you then go back to a Watch window, you will see a hex value such as **0x400** displayed.

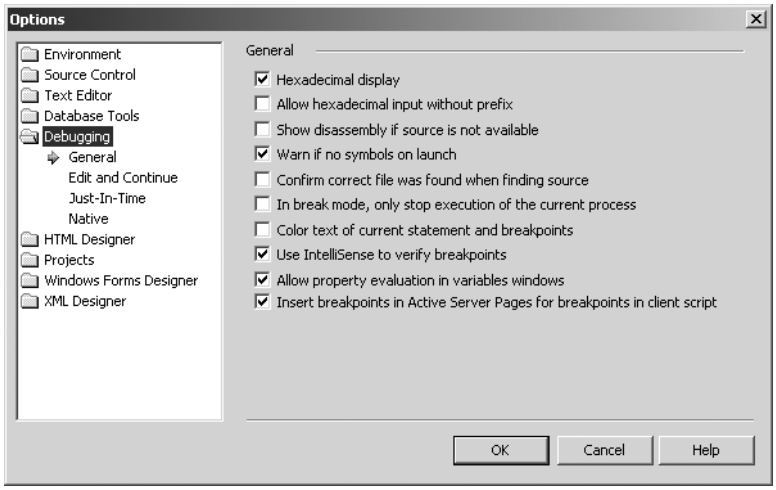


Figure 3–15 *Setting hexadecimal display in Debugging Options.*

SINGLE STEPPING

When you are stopped in the debugger, you can single step. You can also begin execution by single stepping. 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  .

With Step Into you will step into a function, if the cursor is positioned on a call to a function. With Step Over you will step to the next line (or statement or instruction, depending on the selection in the dropdown next to the step buttons ). To illustrate Step Into, build the **Bytes\Step2** project, where the multiplication by 1,024 has been replaced by a function call to the static method **OneK**. Set a breakpoint at the first function call, and then Step Into. The result is illustrated in Figure 3–16. Note the red dot at the breakpoint and the yellow arrow in the function.

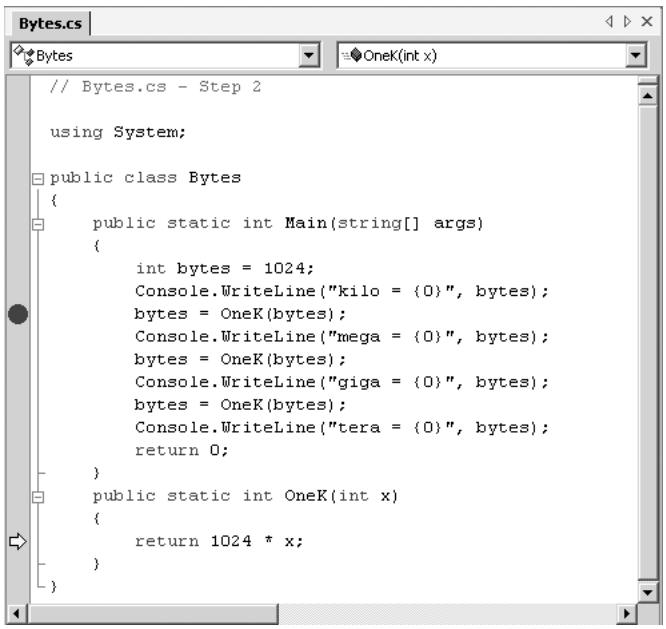


Figure 3–16 *Stepping into a function.*

When debugging, Visual Studio maintains a Call Stack. In our simple example the Call Stack is just two deep. See Figure 3–17.

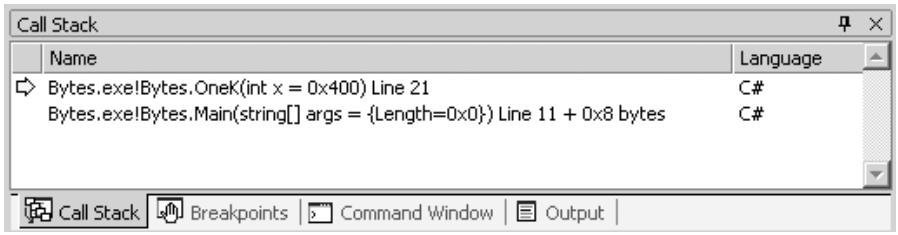


Figure 3–17 *The call stack.*

SUMMARY

Visual Studio.NET is a very rich integrated development environment (IDE), with many features to make programming more enjoyable. In this chapter 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 book. A project can be built in different configurations, such as Debug and Release. Visual Studio.NET has a vast array of features for building database applications, Web applications, components, and many other kinds of projects. It supports many different languages. In this book we are using only a tiny fraction of the capabilities of this powerful tool, but the simple features we employ are very useful, and will certainly make your life as a C# programmer easier.