



# Part **1**

# Fundamental C# Programming

**In this section you will find:**

- ◆ **Chapter 1: Introduction to C#**
- ◆ **Chapter 2: Basic C# Programming**
- ◆ **Chapter 3: Expressions and Operators**
- ◆ **Chapter 4: Decisions, Loops, and Preprocessor Directives**
- ◆ **Chapter 5: Object-Oriented Programming**
- ◆ **Chapter 6: More about Classes and Objects**
- ◆ **Chapter 7: Derived Classes**
- ◆ **Chapter 8: Interfaces**
- ◆ **Chapter 9: Strings, Dates, Times, and Time Spans**
- ◆ **Chapter 10: Arrays and Indexers**
- ◆ **Chapter 11: Collections**
- ◆ **Chapter 12: Delegates and Events**
- ◆ **Chapter 13: Exceptions and Debugging**





## Chapter 1

# Introduction to C#

IN THIS CHAPTER, YOU'LL be introduced to the C# language. You'll see a simple example program that displays the words *Hello World!* on your computer's screen, along with the current date and time.

You'll also learn about Microsoft's Rapid Application Development (RAD) tool, Visual Studio .NET. Visual Studio .NET enables you to develop and run programs in an integrated development environment. This environment uses all the great features of Windows, such as the mouse and intuitive menus, and increases your productivity as a programmer.

In the final sections of this chapter, you'll see how to use the extensive documentation from Microsoft that comes with the .NET Software Development Kit (SDK) and Visual Studio .NET. This documentation goes well beyond the text of this book, and you'll find it invaluable as you become an expert C# programmer.

**NOTE** Before you can develop C# programs, you'll need to install the .NET SDK or Visual Studio .NET. You can download the .NET SDK at <http://msdn.microsoft.com/downloads>. Once you've downloaded the executable file, go ahead and run it. Follow the instructions on the screen to install the .NET SDK on your computer. You can also purchase a copy of Visual Studio .NET from Microsoft at their website.

Featured in this chapter:

- ◆ Building Your First C# Program
- ◆ Learning about Visual Studio .NET
- ◆ Working with the .NET Documentation

## Developing Your First C# Program

Learning a new programming language is sometimes a daunting task. To get you started, you'll begin with a variation on the classic "Hello World" program. This program traditionally starts all programming books—and who are we to argue with tradition?

### THE ORIGINS OF THE “HELLO WORLD” PROGRAM

As far as we know, the tradition of the “Hello World” program being used to start programming books began in the seminal work *The C Programming Language* by Brian Kernighan and Dennis Ritchie (Prentice Hall PTR, 1988). Incidentally, C is one of the languages that C# owes its development to, along with Java and C++.

The following program displays the words *Hello World!* on your computer’s screen. The program will also display the current date and time retrieved from your computer. This program, shown in Listing 1.1, illustrates a few simple tenets of the C# language.

#### LISTING 1.1: THE “HELLO WORLD” PROGRAM

```
/*
   Example1_1.cs: a variation on the classic "Hello World!" program.
   This program displays the words "Hello World!" on the screen,
   along with the current date and time
*/

class Example1_1
{
    public static void Main()
    {
        // display "Hello World!" on the screen
        System.Console.WriteLine("Hello World!");

        // display the current date and time
        System.Console.WriteLine("The current date and time is " +
            System.DateTime.Now);
    }
}
```

This program is contained in a text file named `Example1_1.cs`. This file is known as a *program source file*, or simply a *source file*, because it contains the lines that make up the program. You use a compiler to translate a source file into an executable file that a computer can run; you’ll learn more about this later in the “Compiling a Program” section.

**NOTE** You can download all the source files for the programs featured in this book from the Sybex website at [www.sybex.com](http://www.sybex.com). You’ll find instructions on downloading these files in the introduction of this book. Once you’ve downloaded the files, you can follow along with the examples without having to type in the program listings.

The `Example1_1.cs` source file contains the lines that make up the “Hello World” program. You’ll notice that the extension for the `Example1_1.cs` file is `.cs`—this is the recommended extension for C# source files. Because the file is a text file, you can open and view the `Example1_1.cs` file using a text editor such as Notepad. Go ahead and open the file if you want.

**TIP** You can also edit and save source files using Notepad, although as you develop more complex programs you’ll find that Visual Studio .NET is a much more efficient tool to use. You’ll learn about Visual Studio .NET later in this chapter.

Let’s go through the lines in `Example1_1.cs`. The first four lines are as follows:

```
/*
   Example1_1.cs: a variation on the classic "Hello World!" program.
   This program displays the words "Hello World!" on the screen,
   along with the current date and time
*/
```

The compiler ignores anything placed between the `/*` and `*/` characters. They are comments that we’ve used to inform you what the program does. Later, you’ll see the use of single-line comments that start with two forward slash characters (`//`).

The next two lines start a class using the `class` keyword:

```
class Example1_1
{
```

The open curly brace (`{`) marks the beginning of the `Example1_1` class. Similarly, the close curly brace (`}`), shown at the end of Listing 1.1, marks the end of the `Example1_1` class. As you’ll learn in Chapter 5, “Object-Oriented Programming,” you use a *class* to define a template that contains methods and fields—and you can use this template to create objects of that class.

*Methods* are self-contained units of code that carry out a specific task, and they typically consist of one or more program lines. *Fields* are named storage areas where you can store values. The `Example1_1` class doesn’t contain any fields, but it does contain a method named `Main()`.

**NOTE** Programs typically contain a `Main()` method. This method is run, or called, automatically when you run your program. The exception is a type library, which requires another program to call its functionality and therefore doesn’t require a `Main()` method.

In the next section, we’ll take you through the lines in the `Main()` method.

## Understanding the `Main()` Method

As mentioned, methods typically consist of one or more program lines that carry out the method’s task. The program lines that make up a method begin and end with open and close curly braces, respectively. The `Main()` method in the example “Hello World” program is defined as follows:

```
public static void Main()
{
    // display "Hello World!" on the screen
    System.Console.WriteLine("Hello World!");
```

```

// display the current date and time
System.Console.WriteLine("The current date and time is " +
    System.DateTime.Now);
}

```

The `public` keyword is an access modifier that specifies the level of availability of the method outside of the class; `public` specifies that the `Main()` method is available without restriction and may be called anywhere.

**NOTE** You'll learn more about access modifiers in Chapter 5.

As you'll learn in Chapter 6, "More about Classes and Objects," the `static` keyword indicates that the `Main()` method belongs to the class, rather than any particular object of the class. If we didn't use the `static` keyword when defining the method, we would have to first create an object of the class and then call the method. This may sound a little confusing, but you'll understand exactly what we mean after you've read Chapters 5 and 6.

Methods can return a value to the statement from which they are called. For example, you might want to perform some kind of calculation in a method and return the result of that calculation. However, you may not always want to return a value, and that's what the `void` keyword does. As you can see in the example program, the `void` keyword indicates that the `Main()` method doesn't return a value.

Let's take a look at the program lines contained within the open and close curly brackets; these lines carry out the tasks for the method and are run when the `Main()` method is called. The first program line is as follows:

```
// display "Hello World!" on the screen
```

This line begins with two forward slash characters (`//`). These indicate that the line is a comment. As mentioned, the `/*` and `*/` characters also mark the beginning and end of comments. The difference between these two ways of marking lines as comments is that the `//` characters mark a single line as a comment, whereas the `/*` and `*/` characters mark multiple lines as comments. You'll learn more about comments in Chapter 2, "Basic C# Programming."

The second program line in the `Main()` method is as follows:

```
System.Console.WriteLine("Hello World!");
```

This line calls the `WriteLine()` method. This method displays a line of output on your computer's screen. In the example program, the call to this method displays a line containing the words *Hello World!*

As you'll learn in Chapter 6, *namespaces* separate class declarations, and `System` is a namespace created by Microsoft. The `System` namespace contains a number of useful classes you can use in your programs, and you'll see some of them in this book. The `Console` class is one of the classes in the `System` namespace. The `Console` class contains methods you can use to display output on a computer's screen.

**NOTE** The `Console` class also contains methods you can use to read input from the computer's keyboard, and you'll see how to do that in Chapter 2.

As you can see from the previous line, a period (`.`) separates the `System` namespace, the `Console` class, and the `WriteLine()` method. The period is known as the *dot operator*, and it may be used to

separate the namespace, class, and method parts of a program line. You'll learn more about the dot operator in Chapter 5.

The third line in the `Main()` method is another single line comment:

```
// display the current date and time
```

The fourth line in the `Main()` method displays the current date and time:

```
System.Console.WriteLine("The current date and time is " +
    System.DateTime.Now);
```

As you can see, this line uses `System.DateTime.Now` to display the current date and time. `Now` is a *property* of `DateTime` that returns the current date and time set for the computer on which the program is running. You'll learn all about properties in Chapter 6. In a nutshell, the `Now` property reads the current date and time from your computer. `Now` is a static property, which means you can call it without first creating a `DateTime` object.

The remaining lines in `Example1_1.cs` contain close curly braces that end the `Main()` method and the `Example1_1` class.

## Compiling a Program

A program source file is written in text that you can read. Unfortunately, a computer cannot directly run the instructions contained in that source file, and you must first *compile* that file using a piece of software known as a *compiler*. The compiler reads your program source file and converts the instructions contained in that file into code that a computer may then run, or *execute*. The file produced by the compiler is known as an *executable file*. Once you've compiled your program, you can then run it.

You can compile a program using either the command-line compiler that comes with the .NET SDK, or you can use Visual Studio .NET. In this section, you'll see how to use the command-line version of the compiler to compile the `Example1_1.cs` program. Later in the "Introducing Visual Studio .NET" section you'll see how to use Visual Studio .NET to compile a program.

You run the command-line version of the compiler by entering `csc` in the Command Prompt tool, followed by the name of your program source file. For example, to compile `Example1_1.cs`, you would enter the following command in the Command Prompt tool:

```
csc Example1_1.cs
```

**NOTE** You can also enter one or more options that are then passed to the compiler. These options control things like the name of the executable file produced by the compiler. You can see the full list of options in Appendix B, "C# Compiler Options." You can also view the compiler options by entering `csc /help` in the Command Prompt tool.

If you want to follow along with the examples, go ahead and start the Command Prompt tool by selecting `Start > Programs > Accessories > Command Prompt`.

**NOTE** If you're using Windows XP rather than Windows 2000, you start the Command Prompt tool by selecting `Start > All Programs > Accessories > Command Prompt`.

Next, you need to change directories to where you copied the `Example1_1.cs` file. To do this, you first enter the partition on your hard disk where you saved the file. For example, let's say you saved

the file in the `C#\programs` directory of the C partition of your hard disk. To access the C partition, you enter the following line into the Command Prompt tool, then you press the Enter key:

```
C:
```

Next, to move to the `C#\programs` directory, you enter `cd` followed by `C#\programs`:

```
cd C#\programs
```

To compile `Example1_1.cs` using `csc`, you enter the following command:

```
csc Example1_1.cs
```

Notice that the name of the program source file follows `csc`—it's `Example1_1.cs` in this case.

**WARNING** *If you get an error when running `csc`, you'll need to add the directory where you installed the SDK to your `Path` environment variable. The `Path` environment variable specifies a list of directories that contain executable programs. Whenever you run a program from the Command Prompt tool, the directories in the `Path` variable are searched for the program you want to run. Your current directory is also searched. To set your `Path` environment variable, select `Start > Settings > Control Panel`. Then double-click `System` and select the `Advanced` tab. Next, click the `Environment Variables` button and double-click `Path` from the system variables area at the bottom. Finally, add the directory where you installed the SDK to your `Path` environment variable. Click `OK` to save your change, and then click `OK` again on the next dialog box. Next, restart the Command Prompt tool so that your change is picked up. You should then be able to run `csc` successfully.*

The compiler takes the `Example1_1.cs` file and compiles it into an executable file named `Example1_1.exe`. The `.exe` file contains instructions that a computer can run—and the `.exe` file extension indicates the file is an executable file.

You run an executable file using the Command Prompt tool by entering the name of that executable file. For example, to run the `Example1_1.exe` file, you enter the following line in the Command Prompt tool and then you press the Enter key:

```
Example1_1
```

**NOTE** *You can omit the `.exe` extension when running a program. For example, you can use `Example1_1` to run `Example1_1.exe`.*

When you run the program, you should see the following text displayed in your Command Prompt window:

```
Hello World!  
The current date and time is 8/1/2002 12:22:44 PM
```

Needless to say, your date and time will differ from that shown in the previous line. This date and time is read from your computer when you run the program.

## Introducing the Microsoft Intermediate Language (MSIL)

When you compile a program, the `.exe` file produced by the compiler contains instructions written in Microsoft Intermediate Language (MSIL). MSIL is frequently abbreviated to IL. Now, a computer

can only run programs written in their own native tongue: machine code. *Machine code* is a series of binary numbers (zeros and ones) that a computer can understand and run.

IL instructions are not written in machine code—and therefore an additional step is required to convert the IL into machine code before your program is run for the first time. This step is performed automatically by a piece of software known as the Just In Time (JIT) compiler.

When you run your program, the IL instructions in your `.exe` file are converted by the JIT compiler into machine code that the computer then runs. This is efficient because the JIT compiler detects the type of Central Processing Unit (CPU) in the computer and produces machine code specifically tailored to that CPU. This results in machine code that runs as fast as possible.

**NOTE** *When you distribute your programs, you can then be sure your program will run as fast as possible, regardless of the CPU used by the computer on which your program runs.*

JIT compilation is only performed the first time your program is run, and the resulting machine code is automatically stored. When your program runs again, the stored machine code is reused. That way, the computer doesn't need to keep re-compiling the IL instructions into machine code. Of course, when the computer is turned off or rebooted, the JIT will need to recompile your program into IL instructions when it is run again.

## Introducing Visual Studio .NET

Visual Studio .NET (VS .NET) is Microsoft's Rapid Application Development (RAD) tool. VS .NET is an integrated development environment that you can use to create many types of .NET programs. VS .NET is a more productive tool than a simple text editor such as Notepad. This is because VS .NET allows you to enter your program, compile it, and run it—all within an easy-to-use graphical Windows environment.

VS .NET also enables you to step through each line in your program as it runs. This is useful when your program has errors, or *bugs*. The process of getting rid of bugs in your program is known as *debugging*—and you'll learn about this in Chapter 13, "Exceptions and Debugging." You'll also learn how to use VS .NET's *debugger* in that chapter. You use the debugger to step through each line in your program.

In the previous section, you saw a program that displayed the words *Hello World!* along with the current date and time on your computer's screen. This type of program is known as a *console application* because it displays output directly on the computer's screen on which the program is running.

You can use VS .NET to create console applications, as well as the following type of applications:

**Windows Applications** Windows applications are programs that take advantage of the visual controls offered by the Windows operating system, such as menus, buttons, and editable text boxes. Windows Explorer, which you use to navigate the file system of your computer, is one example of a Windows application. You'll learn about Windows programming in Chapter 24, "Introduction to Windows Applications."

**ASP.NET Applications** ASP.NET applications are programs that run over the Internet. You access an ASP.NET application using a web browser, such as Internet Explorer. Examples of ASP.NET applications would be online banking, stock trading, or auction systems. You'll learn about ASP.NET programming in Chapter 25, "Active Server Pages: ASP.NET!"

**ASP.NET Web Services** ASP.NET web services are also programs that run over the Internet. ASP.NET web services are also known as XML web services. The difference is that you can use them to offer a service that could be used in a distributed system of interconnected services. For example, Microsoft’s Passport web service offers identification and authentication of web users you could then use in your own web application. You’ll learn about web services in Chapter 26, “Web Services.”

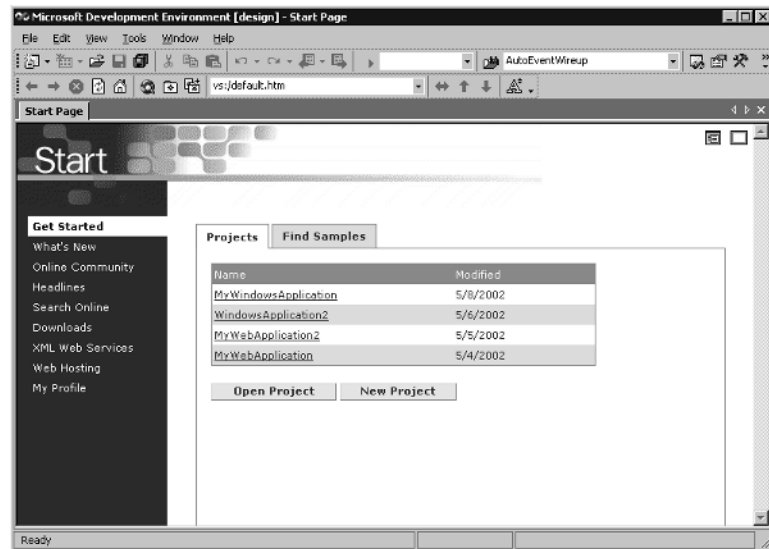
The previous list is not an exhaustive list of the types of applications you can develop with VS .NET, but it does give you flavor for the broad range of VS .NET’s capabilities.

In the rest of this section, you’ll see how to develop and run the “Hello World” program using VS .NET. If you’ve installed VS .NET on your computer, you’ll be able to follow along with the example. If you don’t have VS .NET, then don’t worry—you’ll be able to see what’s going on from the figures provided.

## Starting Visual Studio .NET and Creating a Project

All of your work in VS .NET is organized into *projects*. Projects contain the source and executable files for your program, among other items. If you have VS .NET installed, go ahead and start it by selecting Start > Programs > Microsoft Visual Studio .NET > Microsoft Visual Studio .NET. Once VS .NET has started, you’ll see the Start page (see Figure 1.1).

**FIGURE 1.1**  
The Start page



From the Start page, you can see any existing projects you’ve created. You can open and create projects using the Open Project and New Project buttons, respectively. You’ll create a new project shortly.

## USING THE VS .NET LINKS

As you can see from Figure 1.1, VS .NET contains a number of links on the left of the Start page. Some of these links provide access to useful information on the Internet about .NET; the links are as follows:

**Get Started** Opens the Start page. You can open and create projects from the Start page, and you saw an example Start page earlier in Figure 1.1.

**What's New** Use this link to view any updates for VS .NET or Windows. You can also view upcoming training events and conferences.

**Online Community** Get in touch with other members of the .NET community. Includes links to websites and newsgroups.

**Headlines** View the latest news on .NET.

**Search Online** Use this link to search the MSDN Online Library for technical material such as published articles on .NET.

**Downloads** Download trial applications and example programs from the websites featured here.

**XML Web Services** Find registered XML web services that you can then use in your own programs. XML web services are also known as ASP.NET web services. You'll learn more about web services in Chapter 26.

**Web Hosting** A web hosting company is an organization that can take your program and run it for you. They take care of the computers on which your program runs. You use the Web Hosting link to view companies that provide web hosting services to run your programs.

**My Profile** This link allows you to set items such as your required keyboard scheme and window layout.

Go ahead and click these links and explore the information provided. As you'll see, there's a huge amount of information about .NET on the Internet.

## CREATING A NEW PROJECT

When you're finished examining the information in the previous links, go ahead and create a new project by clicking the New Project button on the Get Started page.

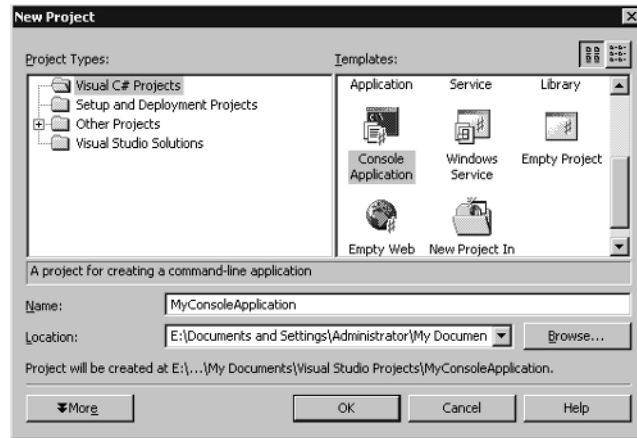
**NOTE** You can also create a new project by selecting *File > New > Project* or by pressing *Ctrl+Shift+N* on your keyboard.

When you create a new project, VS .NET displays the New Project dialog box that you use to select the type of project you want to create. You also enter the name and location of your new project; the location is the directory where you want to store the files for your project.

Because you're going to be creating a C# console application, select Visual C# Projects from the Project Types section on the left of the New Project dialog box, and select Console Application from the Templates section on the right. Enter **MyConsoleApplication** in the Name field, and keep the default directory in the Location field. Figure 1.2 shows the completed New Project dialog box with these settings.

**FIGURE 1.2**

The New Project dialog box with the appropriate settings for a C# console application



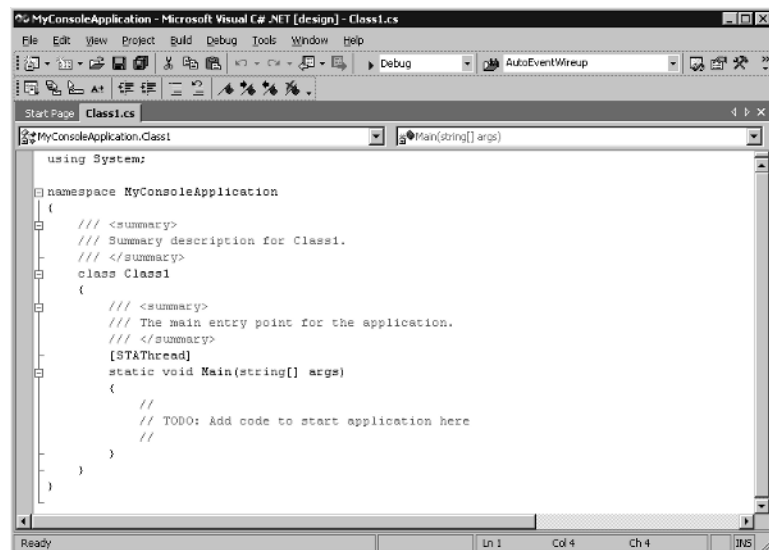
Click the OK button to create the new project.

### WORKING IN THE VS .NET ENVIRONMENT

Once your new project has been created, the main development screen is displayed (see Figure 1.3). This screen is the environment in which you'll develop your project. As you can see, VS .NET has already created some starting code for you; this code is a skeleton for your program—you'll see how to modify it shortly. In this section, we'll give you a brief description of the different parts of the VS .NET environment.

**FIGURE 1.3**

The VS .NET environment



**NOTE** Depending on your settings for VS .NET, your screen may look slightly different from that shown in Figure 1.3.

The VS .NET menu contains the following items:

**File** From the File menu, you can open, close, and save project files.

**Edit** From the Edit menu, you can cut, copy, and paste text from the Clipboard. The Clipboard is a temporary storage area.

**View** From the View menu, you can hide and show different windows such as the Solution Explorer (which allows you to see the files that make up your project), Class View (which allows you to see the classes and objects in your project), Server Explorer (which allows you to explore items such as databases—you’ll learn about databases and the use of Server Explorer in Part III of this book), and the Properties window (which allows you to set the properties of objects, such as the size of a button, for example), among others. You can also use the View menu to select the toolbars you want to display.

**Project** From the Project menu, you can add class files to your project and add Windows forms and controls (you’ll learn about Windows forms and controls in Part III).

**Build** From the Build menu, you can compile the source files in your project.

**Debug** From the Debug menu, you can start your program with or without debugging. Debugging enables you to step through your program line by line looking for errors. You’ll learn about the debugger in Chapter 13.

**Tools** From the Tools menu, you can connect to a database and customize your settings for VS .NET (for example, you can set the colors used for different parts of your program lines or set the initial page displayed by VS .NET when you start it).

**Window** From the Window menu, you can like switch between files you’ve opened and hide windows.

**Help** From the Help menu, you can open the documentation on .NET. You’ll learn how to use this documentation later in this chapter in the “Using the .NET Documentation” section.

The VS .NET toolbar contains a series of buttons that act as shortcuts to some of the options in the menus. For example, you can save a file or all files, cut and paste text from the Clipboard, and start a program using the debugger. You’ll learn how to use some of these features in this chapter.

The code shown in the window (below the toolbar) with the title `Class1.cs` is code that is automatically generated by VS .NET, and in the next section you’ll modify this code.

### MODIFYING THE VS .NET-GENERATED CODE

Once VS .NET has created your project, it will display some starting code for the console application with a class name of `Class1.cs`. You can use this code as the beginning for your own program. Figure 1.3—shown earlier—shows the starting code created by VS .NET.

The `Main()` method created by VS .NET is as follows:

```
static void Main(string[] args)
{
    //
    // TODO: Add code to start application here
    //
}
```

As you can see, this code contains comments that indicate where you add your own code. Go ahead and replace the three lines in the `Main()` method with the lines shown in the following `Main()` method:

```
static void Main(string[] args)
{
    // display "Hello World!" on the screen
    System.Console.WriteLine("Hello World!");

    // display the current date and time
    System.Console.WriteLine("The current date and time is " +
        System.DateTime.Now);
}
```

As you can see, the new lines display the words *Hello World!* on the screen, along with the current date and time. Once you've replaced the code in the `Main()` method, the next steps are to compile and run your program.

## Compiling and Running the Program

As always, you must first compile your program before you can run it. Because programs in VS .NET are organized into projects, you must compile the project—this is also known as *building* the project. To build your project, select **Build > Build Solution**. This compiles the `Class1.cs` source file into an executable file.

**TIP** You can also press *Ctrl+Shift+B* on your keyboard to build your project.

Finally, you can now run your program. To run your program, select **Debug > Start Without Debugging**. When you select **Start Without Debugging**, the program will pause at the end allowing you to view the output.

**TIP** You can also press *Ctrl+F5* on your keyboard to run your program.

When you run your program, VS .NET will run the program in a new Command Prompt window, as shown in Figure 1.4. Your program is run in a Command Prompt window because it is a console application.

To end the program, go ahead and press any key. This will also close the Command Prompt window.

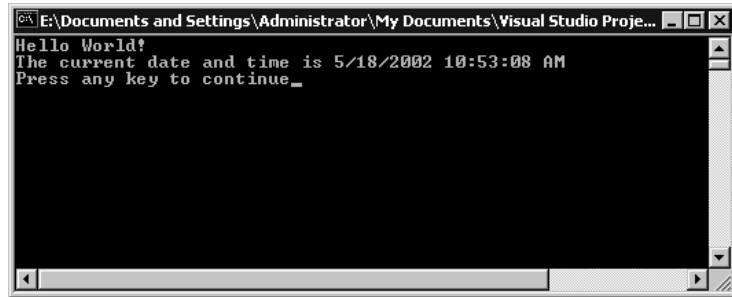
You've barely scratched the surface of VS .NET in this section. You'll explore some of the other features of VS .NET later in this book—including how to step through each line in a program using

the debugger that is integrated with VS .NET. You typically use the debugger to find errors in your programs, and you'll see how to use the debugger in Chapter 13.

In the next section, you'll learn how to use the extensive documentation that comes with .NET.

**FIGURE 1.4**

The running program



## Using the .NET Documentation

Both the .NET SDK and VS .NET come with extensive documentation, including the full reference to all the classes in .NET. As you become proficient with C#, you'll find this reference documentation invaluable.

In the following sections, you'll see how to access the .NET documentation, see how to search the documentation, and view some of the contents of the documentation. Depending on whether you're using the .NET SDK or VS .NET, you access the documentation in a slightly different way. You'll see how to use both ways to access the documentation in this section.

**NOTE** *The documentation that comes with the .NET SDK is a subset of the documentation that comes with VS .NET.*

### Accessing the Documentation Using the .NET SDK

If you're using the .NET SDK, you access the documentation by selecting Start > Programs > .NET Framework SDK > Overview. Figure 1.5 shows the .NET Framework SDK Documentation home page—this is the starting page for the documentation.

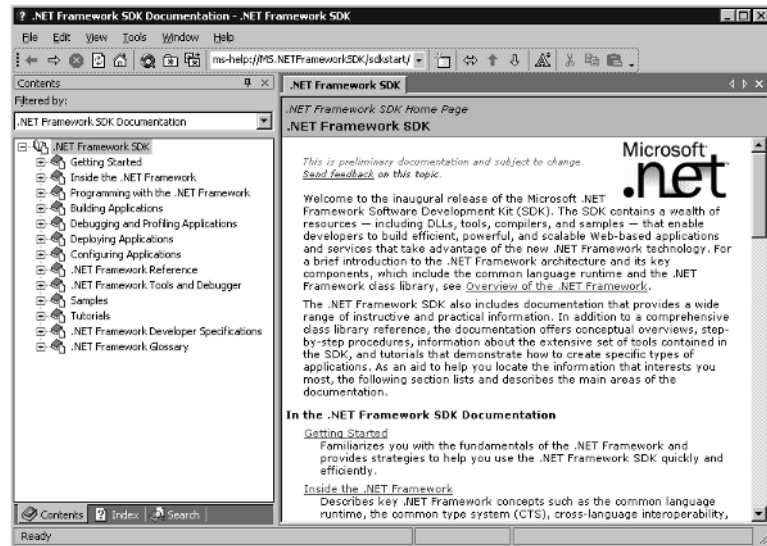
On the left of the page, you can see the various sections that make up the contents of the documentation. You can view the index of the documentation by selecting the Index tab at the bottom of the page.

**TIP** *You can also view the Index window by selecting Help > Index or by pressing Ctrl+Alt+F2 on your keyboard.*

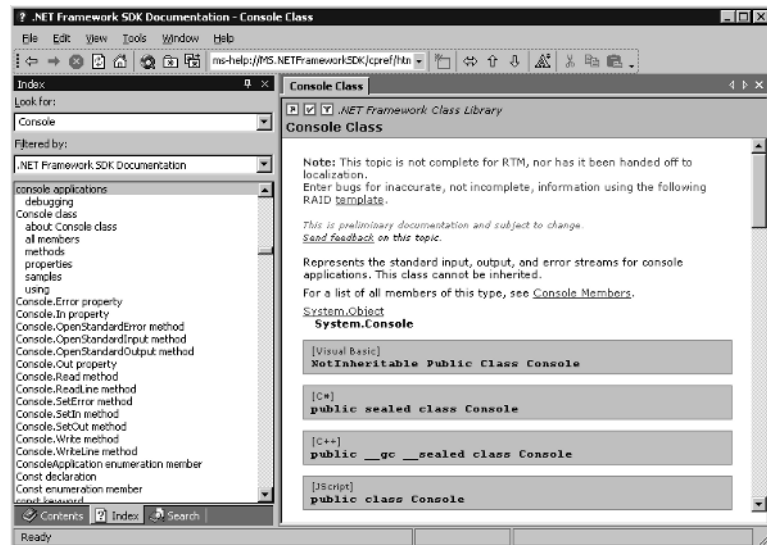
You can search the index by entering a word in the Look For field of the Index tab. Figure 1.6 shows the results of searching for *Console*. Figure 1.6 also shows the overview for the *Console* class on the right. We opened this overview by double-clicking the *About Console Class* link in the Index window on the left of the screen.

**FIGURE 1.5**

The documentation home page

**FIGURE 1.6**

Searching the index for the word *Console*



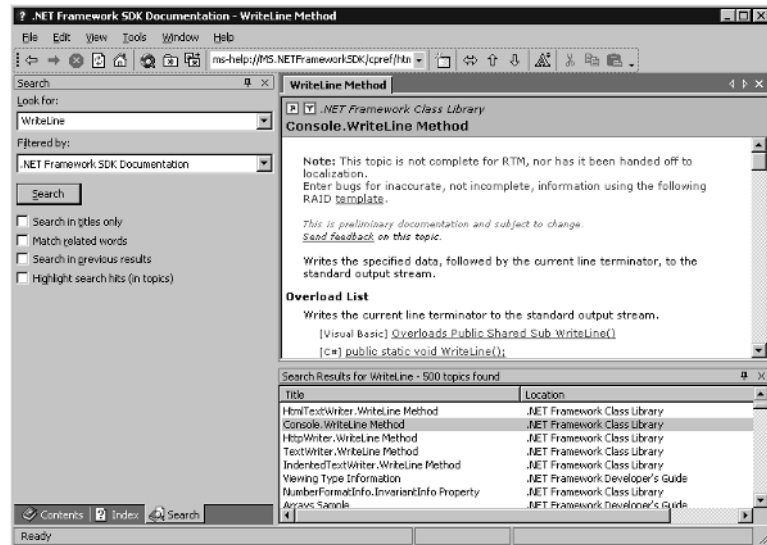
You can also search all pages in the documentation using the Search tab. You display the Search tab by selecting it from the bottom of the screen.

**TIP** You can also view the Search window by selecting **Help** > **Search** or by pressing **Ctrl+Alt+F3** on your keyboard.

You enter the words you want to search for in the Look For field of the Search window. Figure 1.7 shows the Search tab and the search results returned by a search for *WriteLine*. When you run the search, the names of the pages that contain your required words are displayed in the Search Results window that appears at the bottom of the screen (you can see this window in Figure 1.7).

**FIGURE 1.7**

Searching all of the documentation for the word *WriteLine*



*TIP* You can also view the Search Results window by selecting **Help** > **Search results** or by pressing **Shift+Alt+F3** on your keyboard.

You view the contents of a particular page shown in the Search Results window by double-clicking the appropriate line. For example, in Figure 1.7 shown earlier, we double-clicked the second line in the Search Results window. This line contained the “Console.WriteLine Method” page and as you can see, this page is displayed in the window above the search results in Figure 1.7.

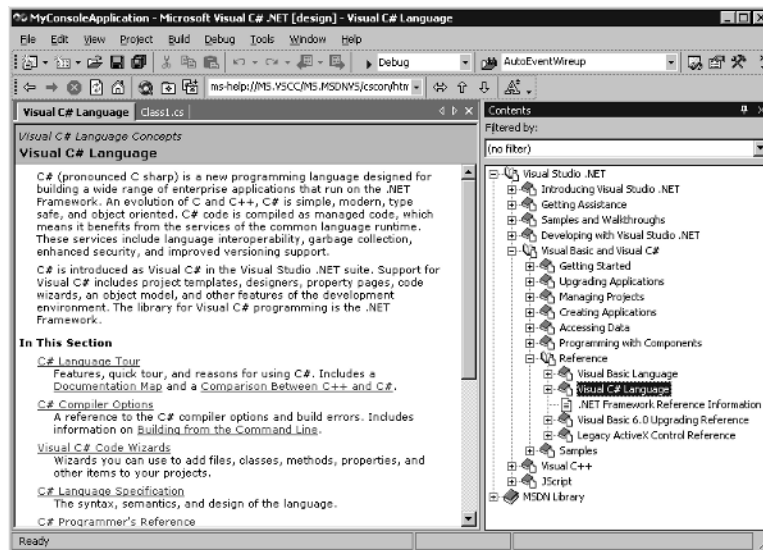
In the next section, you’ll see how to access the documentation using VS .NET.

## Accessing the Documentation Using VS .NET

If you’re using VS .NET, you access the documentation using the Help menu. To access the contents of the documentation, you select **Help** > **Contents**. Figure 1.8 shows the contents displayed in VS .NET. Notice that the documentation is displayed directly in VS .NET, rather than in a separate window as is done when viewing documentation with the .NET SDK.

**FIGURE 1.8**

The documentation contents viewed in VS .NET



**NOTE** The same keyboard shortcuts shown in the previous section also apply to VS .NET.

The Help menu also provides access to similar Index and Search windows as you saw in the previous section.

## Summary

In this chapter, you were introduced to the C# language. You saw a simple example program that displayed the words *Hello World!* on your computer's screen, along with the current date and time. You also learned about the compiler.

You also learned about Microsoft's Rapid Application Development (RAD) tool, Visual Studio .NET. Visual Studio .NET allows you to develop, run, and debug programs in an integrated development environment.

In the final sections of this chapter, you saw how to use the extensive documentation from Microsoft that comes with the .NET SDK and Visual Studio .NET. As you become an expert C# programmer, you'll find this documentation invaluable.

In the next chapter, you'll learn more about the basics of C# programming.