Chapter 1

Creating Your First C# Console Application

In This Chapter

- ▶ A quick introduction to programming
- ▶ Creating a simple console application
- ▶ Reviewing the console application
- ▶ Creating the rest of the programs in this book

In this chapter, I explain a little bit about computers, computer languages, C#, and Visual Studio 2008. Then I take you through the steps for creating a very simple program written in C#.

Getting a Handle on Computer Languages, C#, and .NET

A computer is an amazingly fast, but incredibly stupid servant. Computers will do anything you ask them to (within reason), they do it extremely fast — and they're getting faster all the time.

Unfortunately, computers don't understand anything that resembles a human language. Oh, you may come back at me and say something like, "Hey, my telephone lets me dial my friend by just speaking his name. I know that a tiny computer runs my telephone. So that computer speaks English." But that's a computer *program* that understands English, not the computer itself.

The language that computers really understand is often called *machine language.* It is possible, but extremely difficult and error-prone, for humans to write machine language.

Humans and computers have decided to meet somewhere in the middle. Programmers create programs in a language that is not nearly as free as human speech but a lot more flexible and easy to use than machine language. The languages occupying this middle ground — C#, for example — are called *high-level* computer languages. (*High* is a relative term here.)

What's a program?

What is a program? In a practical sense, a Windows program is an executable file that you can run by double-clicking its icon. For example, the version of Microsoft Word that I'm using to write this book is a program. You call that an *executable program,* or *executable* for short. The names of executable program files generally end with the extension `.EXE`. Word, for example, is called Winword.exe.

But a program is something else, as well. An executable program consists of one or more *source files.* A C# program file is a text file that contains a sequence of C# commands, which fit together according to the laws of C# grammar. This file is known as a *source file,* probably because it's a source of frustration and anxiety.

Uh, grammar? There's going to be grammar? Just the C# kind, which is much easier than the kind most of us struggled with in junior high school.

What's C#?

The C# programming language is one of those intermediate languages that programmers use to create executable programs. C# combines the range of the powerful-but-complicated C++ with the ease of use of the friendly but more verbose Visual Basic. (Visual Basic's newer .NET incarnation is almost on par with C# in most respects. As the flagship language of .NET, C# tends to introduce most new features first.) A C# program file carries the extension `.CS`.

WARNING!

Some wags have pointed out that C-sharp and D-flat are the same note, but you should not refer to this new language as "D-flat" within earshot of Redmond, Washington.

C# is

- **Flexible:** C# programs can execute on the current machine, or they can be transmitted over the Web and executed on some distant computer.
- **Powerful:** C# has essentially the same command set as C++, but with the rough edges filed smooth.

- **Easier to use:** C# error-proofs the commands responsible for most C++ errors so you spend far less time chasing down those errors.
- **Visually oriented:** The .NET code library that C# uses for many of its capabilities provides the help needed to readily create complicated display frames with drop-down lists, tabbed windows, grouped buttons, scroll bars, and background images, to name just a few.
- **Internet-friendly:** C# plays a pivotal role in the .NET Framework, Microsoft's current approach to programming for Windows, the Internet, and beyond.

  .NET is pronounced *dot net*.
- **Secure:** Any language intended for use on the Internet must include serious security to protect against malevolent hackers.

Finally, C# is an integral part of .NET.

Because this book focuses on the C# language, it's not a Web-programming book, a database book, or a Windows graphical programming book.

What's .NET?

.NET began a few years ago as Microsoft's strategy to open up the Web to mere mortals like you and me. Today it's bigger than that, encompassing everything Microsoft does. In particular, it's the new way to program for Windows. It also gives a C-based language, C#, the simple, visual tools that made Visual Basic so popular. A little background will help you see the roots of C# and .NET.

Internet programming was traditionally very difficult in older languages such as C and C++. Sun Microsystems responded to that problem by creating the Java programming language. To create Java, Sun took the grammar of C++, made it a lot more user-friendly, and centered it around distributed development.

When programmers say "*distributed*," they're describing geographically dispersed computers running programs that talk to each other — in many cases, via the Internet.

When Microsoft licensed Java some years ago, it ran into legal difficulties with Sun over changes it wanted to make to the language. As a result, Microsoft more or less gave up on Java and started looking for ways to compete with it.

Being forced out of Java was just as well because Java has a serious problem: Although Java is a capable language, you pretty much have to write your entire program *in* Java to get the full benefit. Microsoft had too many developers and too many millions of lines of existing source code, so Microsoft had to come up with some way to support multiple languages. Enter .NET.

.NET is a framework, in many ways similar to Java's libraries — and the C# language is highly similar to the Java language. Just as *Java* is both the language itself and its extensive code library, *C#* is really much more than just the keywords and syntax of the C# language. It's those things empowered by a thoroughly object-oriented library containing thousands of code elements that simplify doing about any kind of programming you can imagine, from Web-based databases to cryptography to the humble Windows dialog box.

Microsoft would claim that .NET is much superior to Sun's suite of Web tools based on Java, but that's not the point. Unlike Java, .NET does not require you to rewrite existing programs. A Visual Basic programmer can add just a few lines to make an existing program "Web-knowledgeable" (meaning that it knows how to get data off the Internet). .NET supports all the common Microsoft languages — and more than 40 other languages written by third-party vendors (see `dotnetpowered.com/languages.aspx` for the latest list). However, C# is the flagship language of the .NET fleet. C# is always the first language to access every new feature of .NET.

What is Visual Studio 2008? What about Visual C#?

(You sure ask lots of questions.) The first "Visual" language from Microsoft was Visual Basic. The first popular C-based language from Microsoft was Visual C++. Like Visual Basic, it was called "Visual" because it had a built-in graphical user interface (GUI — pronounced *gooey*). This GUI included everything you needed to develop nifty-giffy C++ programs.

Eventually, Microsoft rolled all its languages into a single environment — Visual Studio. As Visual Studio 6.0 started getting a little long in the tooth, developers anxiously awaited Version 7. Shortly before its release, however, Microsoft decided to rename it Visual Studio .NET to highlight this new environment's relationship to .NET.

That sounded like a marketing ploy to me — until I started delving into it. Visual Studio .NET differed quite a bit from its predecessors — enough to warrant a new name. Visual Studio 2008 is the third-generation successor to the original Visual Studio .NET. (See Bonus Chapter 6 on the Web site for a tour of some of Visual Studio's more potent features.)

Microsoft calls its implementation of the language Visual C#. In reality, Visual C# is nothing more than the C# component of Visual Studio. C# is C#, with or without the Visual Studio.

Okay, that's it. No more questions. (For now, anyway.)

Creating Your First Console Application

Visual Studio 2008 includes an Application Wizard that builds template programs and saves you a lot of the dirty work you'd have to do if you did everything from scratch. (I don't recommend the from-scratch approach.)

Typically, starter programs don't actually do anything — at least, not anything useful (sounds like most of my programs). However, they do get you beyond that initial hurdle of getting started. Some starter programs are reasonably sophisticated. In fact, you'll be amazed at how much capability the App Wizard can build on its own, especially for graphical programs.

The following instructions are for Visual Studio. If you use anything other than Visual Studio, you have to refer to the documentation that came with your environment. Alternatively, you can just type the source code directly into your C# environment. See the introduction to this book for some alternatives to Visual Studio.

Creating the source program

To start Visual Studio, choose Start➪All Programs➪Microsoft Visual Studio 2008➪Microsoft Visual Studio 2008.

Complete these steps to create your C# console app:

1. **Choose File➪New➪Project to create a new project, as shown in Figure 1-1.**

   Visual Studio presents you with lots of icons representing the different types of applications you can create, as shown in Figure 1-2.

2. **From this New Project window, click the Console Application icon.**

   Make sure that you select Visual C# — and under it, Windows — in the Project Types pane; otherwise Visual Studio may create something awful like a Visual Basic or Visual C++ application. Then click the Console Application icon in the Templates pane.

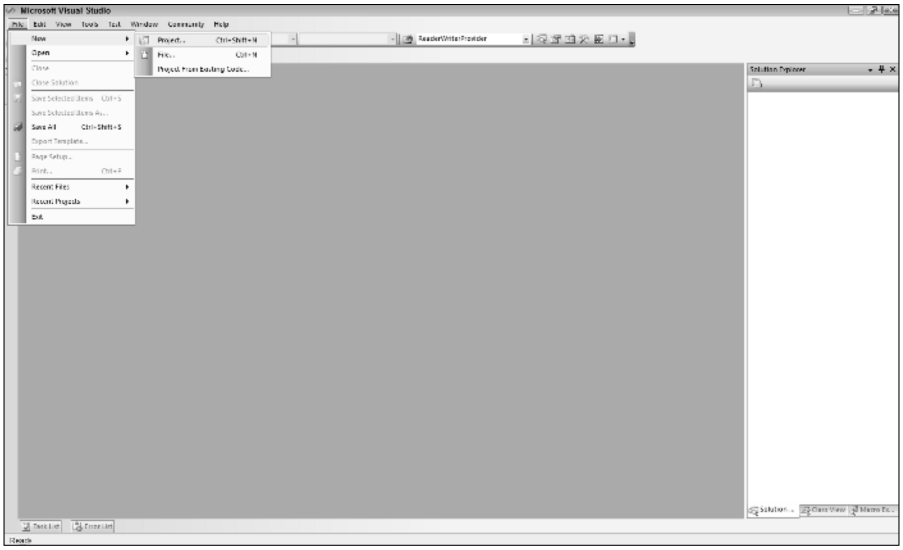
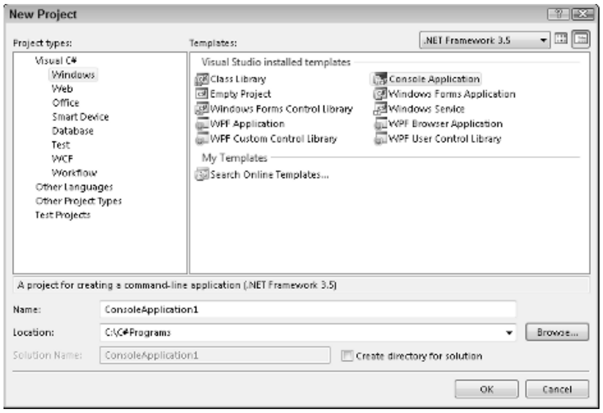


Figure 1-1: Creating a new project starts you down the road to a better Windows application.

Visual Studio requires you to create a project before you can start to enter your C# program. A *project* is like a bucket in which you throw all the files that go into making your program. When you tell your compiler to build (*compile*) the program, it sorts through the project to find the files it needs in order to re-create the executable program.

The default name for your first application is `ConsoleApplication1`, but change it this time to `Program1`.



A project for creating a command-line application (.NET Framework 3.5)

Name: ConsoleApplication1
Location: C:\C#Programs
Solution Name: ConsoleApplication1
Create directory for solution

Figure 1-2: The Visual Studio App Wizard is eager to create a new program for you.

The default place to store this file is somewhere deep in your `Documents` directory. Maybe because I'm difficult (or maybe because I'm writing a book), I like to put my programs where I want them to go, not necessarily where Visual Studio wants them. 'To simplify working with this book, you can change the default program location. Follow these steps to make that happen:

a. *Choose Tools➪Options➪Projects and Solutions➪General.*

b. *Select the new location (I recommend `C:\C#Programs` for this book) in the Visual Studio Projects Location box, and click OK.*

   You can create the new directory in the Project Location dialog box at the same time. Click the folder icon with a small sunburst at the top of the dialog box. (The directory may already exist if you've installed the example programs from the Web site.)

Leave the other boxes in the project settings alone.

3. **Click the OK button.**

   After a bit of disk whirring and chattering, Visual Studio generates a file called `Program.cs`. (If you look in the window labeled Solution Explorer, you see some other files; ignore them for now. If Solution Explorer isn't visible, choose View➪Solution Explorer.) C# source files carry the extension `.CS`. The name `Program` is the default name assigned for the program file.

The contents of your first console app appear as follows:

```
using ...

namespace Program1
{
  class Program
  {
    static void Main(string[] args)
    {

    }
  }
}
```

Along the left edge of the code window, you see several small plus (+) and minus (–) signs in boxes. Click the + sign next to `using....` This expands a *code region,* a handy Visual Studio feature that keeps down the clutter. Here are the directives when you expand the region in the default console app:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

Regions help you focus on the code you're working on by hiding code that you aren't. Certain blocks of code — such as the `namespace` block, `class` block, methods, and other code items — get a `+/-` automatically without a `#region` directive. You can add your own collapsible regions, if you like, by typing **#region** above a code section and **#endregion** after it. It helps to supply a name for the region, such as `Public methods`.' Here's what this code section looks like:

```
#region Public methods
... your code
#endregion Public methods
```

REMEMBER

This name can include spaces. Also, you can nest one region inside another, but regions can't overlap.

For now, `using System;` is the only `using` *directive* you really need. You can delete the others; the compiler lets you know whether you're missing one.

Taking it out for a test drive

To convert your C# program into an executable program, choose Build➪Build `Program1`. Visual Studio responds with the following message:

```
- Build started: Project: Program1, Configuration: Debug Any CPU -

Csc.exe /noconfig /nowarn ... (and much more)

Compile complete -- 0 errors, 0 warnings
Program1 -> C:\C#Programs\ ... (and more)==Build: 1 succeeded or up-to-date, 0
              failed, 0 skipped==
```

TIP

The key point here is the `1 succeeded` part on the last line.

As a general rule of programming, "`succeeded`" is good; "`failed`" is bad.

To execute the program, choose Debug➪Start. The program brings up a black console window and terminates immediately. The program has seemingly done nothing. In fact, this is the case. The template is nothing but an empty shell.

TIP

An alternative command, Debug➪Start Without Debugging, behaves a bit better at this point. Try it out.

Making Your Console App Do Something

Edit the `Program.cs` template file until it appears as follows:

ON THE WEB

```
using System;

namespace Program1
{
  public class Program
  {
    // This is where your program starts.
    static void Main(string[] args)
    {
      // Prompt user to enter a name.
      Console.WriteLine("Enter your name, please:");

      // Now read the name entered.
      string name = Console.ReadLine();

      // Greet the user with the name that was entered.
      Console.WriteLine("Hello, " + name);

      // Wait for user to acknowledge the results.
      Console.WriteLine("Press Enter to terminate...");
      Console.Read();
    }
  }
}
```

TIP

Don’t sweat the stuff following the double or triple slashes (`//` or `///`), and don’t worry about whether to enter one or two spaces or one or two new lines. However, do pay attention to capitalization.

Choose Build➪Build `Program1` to convert this new version of `Program.cs` into the `Program1.exe` program.

From within Visual Studio 2008, choose Debug➪Start Without Debugging. The black console window appears and prompts you for your name. (You may need to activate the console window by clicking it.) Then the window shows `Hello`, followed by the name entered, and displays `Press Enter to terminate . . . .` Pressing Enter closes the window.

You can also execute the program from the DOS command line. To do so, open a Command Prompt window and enter the following:

```
CD \C#Programs\Program1\bin\Debug
```

Now enter **Program1** to execute the program. The output should be identical to what you saw earlier. You can also navigate to the `\C#Programs\Program1\bin\Debug` folder in Windows Explorer and then double-click the `Program1.exe` file.

To open a Command Prompt window, try choosing Tools➪Command Prompt. If that command isn't available on your Visual Studio Tools menu, choose Start➪All Programs➪Microsoft Visual Studio 2008➪Visual Studio Tools➪Visual Studio 2008 Command Prompt.

Reviewing Your Console Application

In the following sections, you take this first C# console app apart one section at a time to understand how it works.

The program framework

The basic framework for all console applications starts as the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Program1
{
  public class Program
  {
    // This is where your program starts.
    public static void Main(string[] args)
    {
       // Your code goes here.
    }
  }
}
```

The program starts executing right after the statement containing `Main()` and ends at the closed curly brace following `Main()`. (I explain the meaning of these statements in due course. More than that I cannot say for now.)

The list of `using` directives can come immediately before or immediately after the phrase `namespace Program1 {`. The order doesn't matter. You can apply `using` to lots of things in .NET. The whole business of namespaces and `using` is explained in Bonus Chapter 1 on the Web site.

Comments

The template already has lots of lines, and I've added several other lines, such as the following (in boldface):

```
// This is where your program starts.
public static void Main(string[] args)
```

C# ignores the first line in this example. This line is known as a *comment.*

Any line that begins with `//` or `///` is free text and is ignored by C#. Consider `//` and `///` to be equivalent for now.

Why include lines if the computer ignores them? Because comments explain your C# statements. A program, even in C#, isn't easy to understand. Remember that a programming language is a compromise between what computers understand and what humans understand. These comments are useful while you write the code, and they're especially helpful to the poor sap — possibly you — who tries to re-create your logic a year later. Comments make the job much easier.

Comment early and often.

The meat of the program

The real core of this program is embedded within the block of code marked with `Main()`, as follows:

```
// Prompt user to enter a name.
Console.WriteLine("Enter your name, please:");

// Now read the name entered.
string name = Console.ReadLine();

// Greet the user with the name that was entered.
Console.WriteLine("Hello, " + name);
```

TIP

Save a ton of routine typing with the new C# Code Snippets feature. Snippets are great for common statements like `Console.WriteLine`. Press Ctrl+K and then Ctrl+X to see a pop-up menu of snippets. (You may need to press Tab once or twice to open up the Visual C# folder or other folders on that menu.) Scroll down the menu to `cw` and press Enter. Visual Studio inserts the body of a `Console.WriteLine()` statement with the insertion point between the parentheses, ready to go. When you have a few of the shortcuts like `cw`, `for`, and `if` memorized, use the even quicker technique: Type **cw** and press Tab twice. (Also try selecting some lines of code, pressing Ctrl+K, and then pressing Ctrl+S. Choose something like `if`. An `if` statement *surrounds* the selected code lines.) The program begins executing with the first C# statement: `Console.WriteLine`. This command writes the character string `Enter your name, please:` to the console.

The next statement reads in the user's answer and stores it in a *variable* (a kind of "workbox") called `name`. (See Chapter 2 for more on these storage locations.) The last line combines the string `Hello,` with the user's name and outputs the result to the console.

The final three lines cause the computer to wait for the user to press Enter before proceeding. These lines ensure that the user has time to read the output before the program continues, as follows:

```
// Wait for user to acknowledge the results.
Console.WriteLine("Press Enter to terminate...");
Console.Read();
```

This step can be important depending on how you execute the program and depending on the environment. In particular, running your console app inside Visual Studio, or from Windows Explorer, makes the lines above necessary — otherwise, the console window closes so fast you can't read the output. If you open a console window and run the program from there, the window stays open regardless.

Introducing the Toolbox Trick

Actually, the key part of the program you've created in the preceding section is the final two lines of code:

```
// Wait for user to acknowledge the results.
Console.WriteLine("Press Enter to terminate...");
Console.Read();
```

The easiest way to recreate those key lines in each future console application that you write is as follows.

Saving code in the Toolbox

The first step is to save those lines in a handy location for future use in a handy place: the Toolbox window. 'With your `Program1` console application open in Visual Studio, follow these steps:

1. **In the `Main()` method of class `Program`, select the lines you want to save — in this case, the lines above.**
2. **Make sure the Toolbox window is open. (If it isn't, open it by choosing View➪Toolbox.)**
3. **Drag the selected lines into the General tab of the Toolbox window and drop them. (Or copy the lines and paste them into the Toolbox.)**

   The Toolbox stores the lines there for you in perpetuity. Figure 1-3 shows the lines placed in the Toolbox.

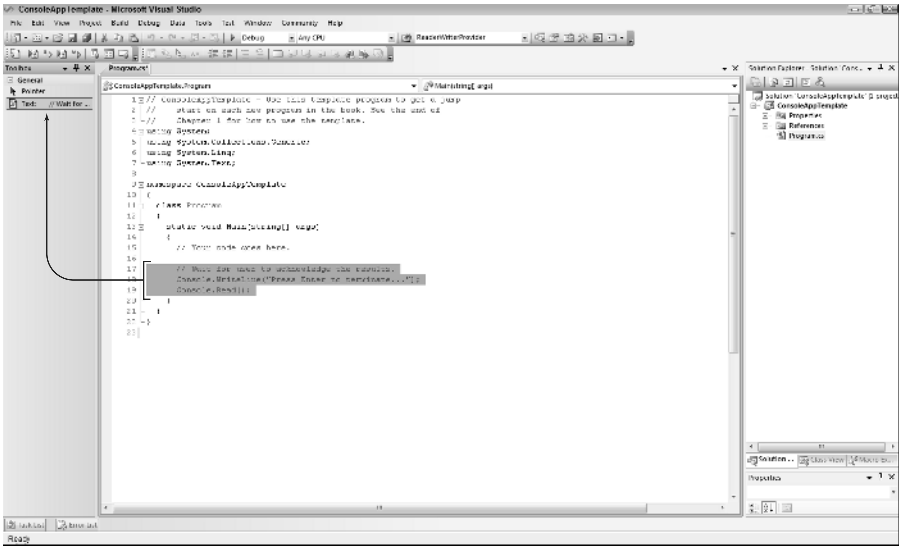
Reusing code from the Toolbox

Now that you have your template text stored in the Toolbox, you can reuse it in all of the console applications you write henceforth. Here's how to use it:

1. In Visual Studio, create a new console application as described earlier in this chapter.
2. Click in the editor at the spot where you'd like to insert some Toolbox text.
3. With the `Program.cs` file open for editing, make sure the Toolbox window is open. (If it isn't, see the procedure above.)
4. In the General tab of the Toolbox window (other tabs could be showing), find the saved text you want to use and double-click it.

   The selected item is inserted at the insertion point in the editor window.

With that boilerplate text in place, you can write the rest of your application above those lines. That's it. You now have a finished console app. Try it out for oh, say, 30 seconds. Then head for Chapter 2.


ConsoleAppTemplate - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
ReaderWriterProvider
Toolbox
General
Pointer
Text: // Wait for ...
Program.cs*
ConsoleAppTemplate.Program
Main(string[] args)
Solution Explorer - Solution Cons...
Solution 'ConsoleAppTemplate' (1 project)
ConsoleAppTemplate
Properties
References
Program.cs
Properties
Task List
Error List
Ready

```
// ConsoleAppTemplate - Use this template program to get a jump
//     start on each new program in the book. See the end of
//     Chapter 1 for how to use the template.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleAppTemplate
{
  class Program
  {
    static void Main(string[] args)
    {
      // Your code goes here.

      // Wait for user to acknowledge the results.
      Console.WriteLine("Press Enter to terminate...");
      Console.Read();
    }
  }
}
```

Figure 1-3: Setting up the Toolbox with some handy saved text for future use.