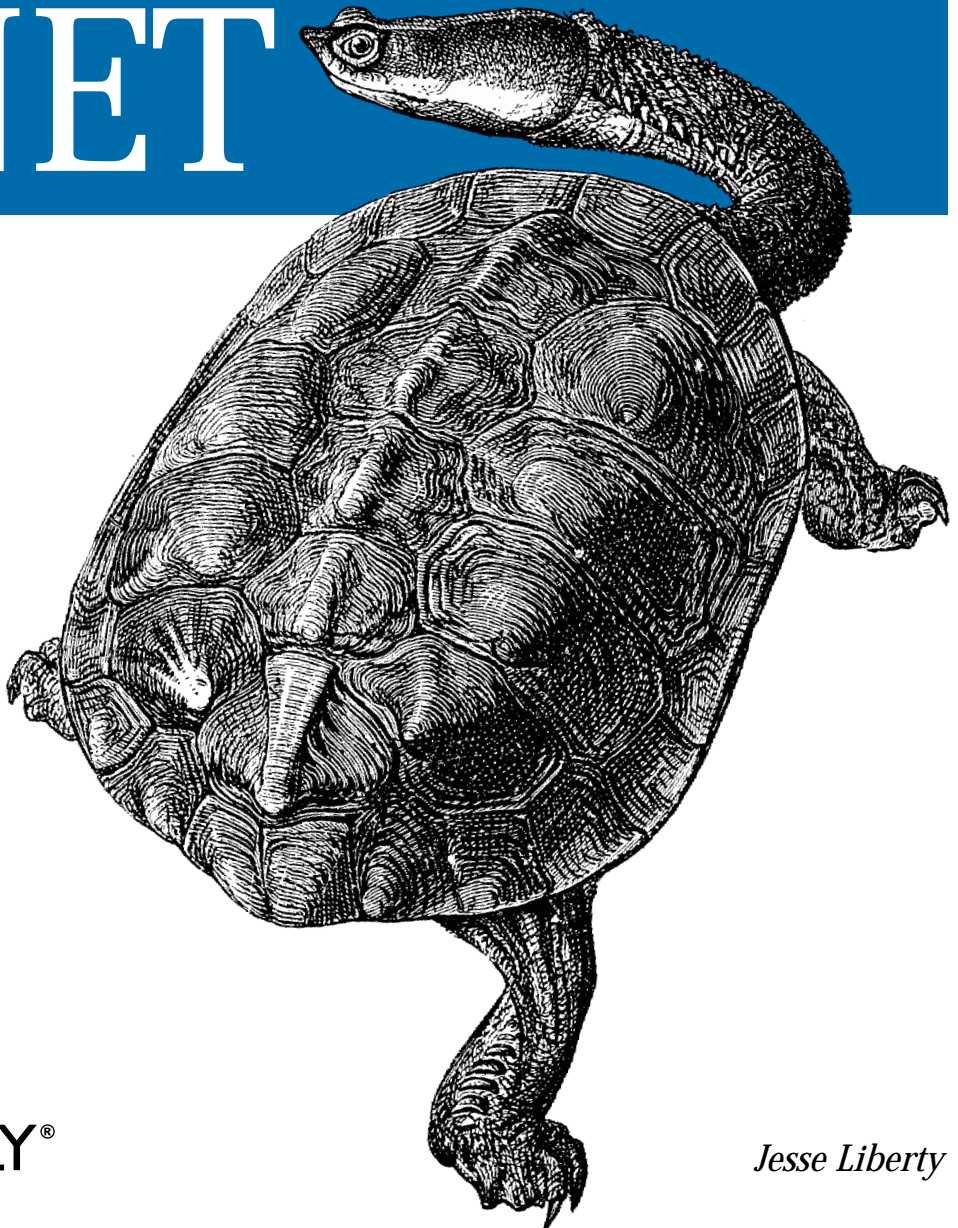


*Language, Object-Oriented Programming  
& Visual Studio .NET*

*Learning*

# Visual Basic .NET



O'REILLY®

*Jesse Liberty*

---

# Learning Visual Basic .NET

*Jesse Liberty*

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

**O'REILLY®**

# Getting Started with VB.NET

You can use VB.NET to create three different types of programs:

- Web applications
- Windows applications
- Console applications

The .NET platform is web-centric. The VB.NET language was developed to allow .NET programmers to create very large, powerful, high-quality web applications quickly and easily. The .NET technology for creating web applications is called ASP.NET.

ASP.NET, the next generation from ASP (Active Server Pages), is composed of two Microsoft development technologies: Web Forms and Web Services. While the development of fully realized web applications using these technologies is beyond the scope of this book, learning the basics of the VB.NET language will certainly get you started in the right direction. VB.NET is generally acknowledged to be one of the languages of choice for ASP.NET development.

Typically, you'll create an ASP.NET application when you want your program to be available to end users on any platform (e.g., Windows, Mac, Unix). By serving your application over the Web, end users can access your program with any browser.

When you want the richness and power of a native application running directly on the Windows platform, alternatively you might create a desktop-bound Windows application. The .NET tools for building Windows applications are called Windows Forms; a detailed analysis of this technology is also beyond the scope of this book.

However, if you don't need a Graphical User Interface (GUI) and just want to write a simple application that talks to a console window (i.e., what we used to call a DOS box), you might consider creating a console application. This book makes extensive use of console applications to illustrate the basics of the VB.NET language.

Web, Windows, and console applications are described and illustrated in the following pages.

### *Console applications*

A console application runs in a console window, as shown in Figure 2-1. A console window (or DOS box) provides simple text-based output.

Console applications are very helpful when learning a language because they strip away the distraction of the Graphical User Interface. Rather than spending your time creating complex windowing applications, you can focus on the details of the language constructs, such as how you create classes and methods, how you branch based on runtime conditions, and how you loop. All these topics will be covered in detail in coming chapters.

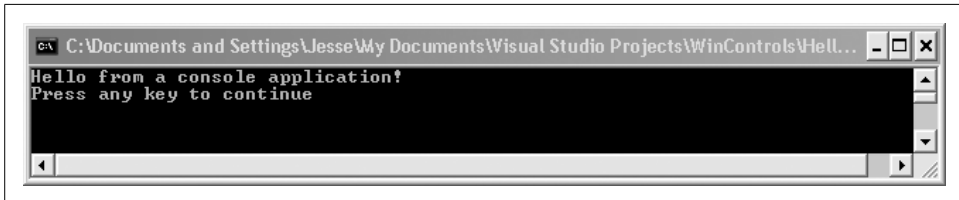


Figure 2-1. A console application

### *Windows applications*

A Windows application runs on a PC's desktop. You are already familiar with Windows applications such as Microsoft Word or Excel. Windows applications are much more complex than console applications and can take advantage of the full suite of menus, controls, and other widgets you've come to expect in a modern desktop application. Figure 2-2 shows the output of a simple windows application.



Figure 2-2. A Windows application

## ASP.NET applications

An ASP.NET application runs on a web server and delivers its functionality through a browser, typically over the Web. ASP.NET technology facilitates developing web applications quickly and easily. Figure 2-3 shows a message from a simple ASP.NET application.

Although most commercial applications will be either Windows or ASP.NET programs, console applications have a tremendous advantage in a VB.NET primer. Windows and ASP.NET applications bring a lot more overhead; there is great complexity in managing the window and all the events associated with the window. (Events are covered in Chapter 18.) Console applications keep things simple, allowing you to focus on the features of the language.

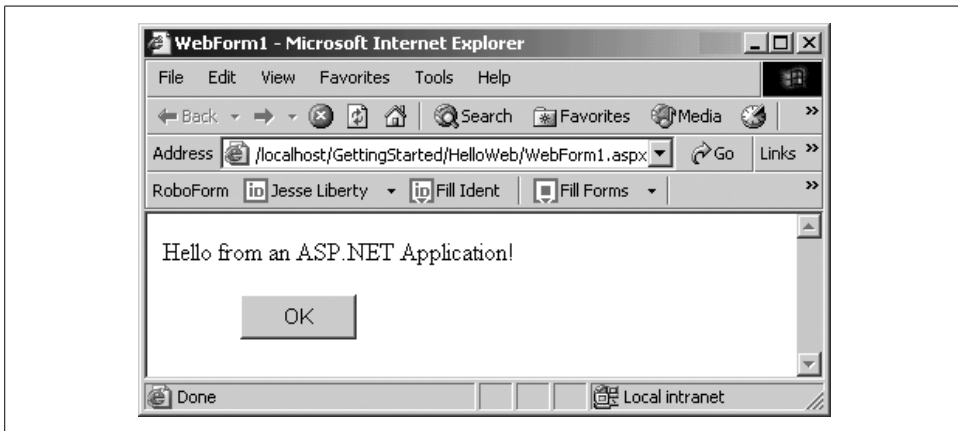


Figure 2-3. An ASP.NET application



This book does not ever go into all the myriad details of building robust Windows and ASP.NET applications. For complete coverage of these topics, please see *Programming ASP.NET* and *Programming .NET Windows Applications* both by Jesse Liberty and Dan Hurwitz (O'Reilly).

## What's in a Program?

A program consists of English-language instructions called *source code*. The syntax for these instructions is strictly defined by the language. Source code consists of a series of statements. A statement is an instruction to the compiler. Each instruction must be formed correctly, and one task you'll face when learning VB.NET will be to learn the correct syntax of the language. For example, in VB.NET every statement ends with a carriage return or linefeed.

Each instruction has a semantic meaning that expresses what it is you are trying to accomplish. Although you must follow the syntax, the semantics of the language are far more important in developing effective object-oriented programs. This book will provide insight into both the syntax and the semantics of good VB.NET programs.

You will save the source code you write in a text file. You can write this source code file using any simple text editor (such as Notepad), or you can use the Visual Studio .NET Integrated Development Environment (IDE). Visual Studio .NET is described in Chapter 4.

Once you write your program, you compile it using the VB.NET compiler. The end result of compiling the program is an application.

## Your First Program: Hello World

In this chapter, you will create a very simple application that does nothing more than display the words “Hello World” to your monitor. This basic console application is the traditional first program for learning any new language; it demonstrates some of the basic elements of a VB.NET program.

Once you write your “Hello World” program and compile it, this chapter will provide a line-by-line analysis of the source code. This analysis gives something of a preview of the language, the fundamentals of which are described much more fully in Chapter 5.

As explained earlier, you can create VB.NET programs with any text editor. You can, for example, create each of the three programs shown previously (in Figure 2-1, Figure 2-2, and Figure 2-3) with Notepad. To demonstrate that this is possible, you’ll write your very first VB.NET program using Notepad.

Begin by opening Notepad and typing in the program exactly as shown in Example 2-1.

*Example 2-1. Hello World in Notepad*

```
Module HelloWorld
    ' every console app starts with Main
    Sub Main()
        System.Console.WriteLine("Hello world!")
    End Sub
End Module
```

That is the entire program. Save it to your disk as a file called *helloworld.vb*.

We’ll examine this program in some detail in just a moment. First, however, it must be compiled.

## The Compiler

Once you save your program to disk, you must compile the code to create your application. Compiling your source code means running a compiler and passing in the source code file. You run the compiler by opening a command prompt (DOS box) and entering the program name *vbc*. Then you “pass in” your source code file by entering the filename on the command line, as in the following:

```
vbc helloworld.vb
```

The job of the compiler is to turn your source code into a working program. It turns out to be just slightly more complicated than that, because .NET uses an intermediate language called Microsoft Intermediate Language (MSIL, sometimes abbreviated to IL). The compiler reads your source code and produces IL. The .NET Just In Time (JIT) compiler then reads your IL code and produces an executable application in memory.

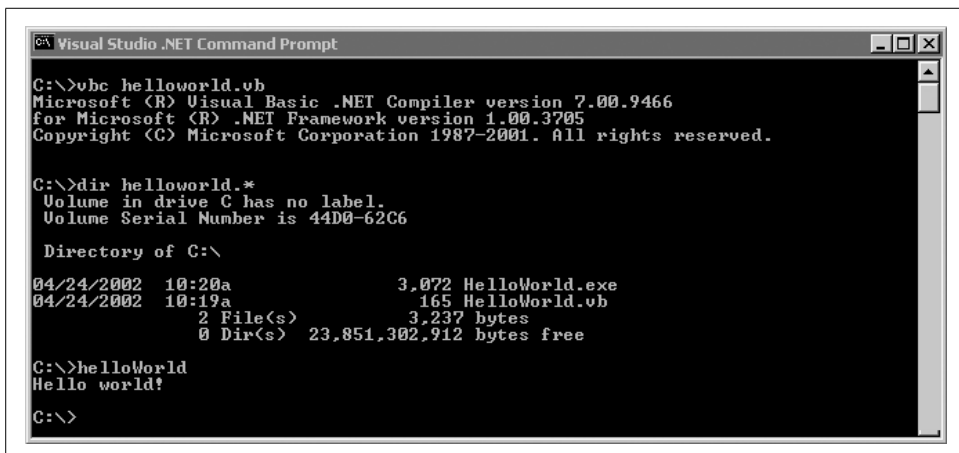
Microsoft provides a command window (through Visual Studio .NET) with the correct environment variables set. Open a command window by selecting the following menu items in this order:

```
Start -> Programs -> Microsoft Visual Studio .NET  
-> Visual Studio.NET Tools -> Visual Studio .NET Command Prompt
```

Then navigate to the directory in which you created your code file and enter the following command:

```
vbc helloworld.vb
```

The Microsoft VB.NET compiler compiles your code; when you display the directory you’ll find the compiler has produced an executable file called *helloworld.exe*. Type *helloworld* at the command prompt, and your program will execute, as shown in Figure 2-4.



```
Visual Studio .NET Command Prompt  
C:\>vbc helloworld.vb  
Microsoft (R) Visual Basic .NET Compiler version 7.00.9466  
for Microsoft (R) .NET Framework version 1.00.3705  
Copyright (C) Microsoft Corporation 1987-2001. All rights reserved.  
  
C:\>dir helloworld.*  
Volume in drive C has no label.  
Volume Serial Number is 44D0-62C6  
  
Directory of C:\  
04/24/2002  10:20a                3,072 HelloWorld.exe  
04/24/2002  10:19a                 165 HelloWorld.vb  
                2 File(s)                3,237 bytes  
                0 Dir(s)      23,851,302,912 bytes free  
  
C:\>helloWorld  
Hello world!  
C:\>
```

Figure 2-4. Compiling and running Hello World

Presto! You are a VB.NET programmer. That's it, close the book, you've done it. Okay, don't close the book; there are details to examine, but take a moment to congratulate yourself. Have a cookie.

Granted, the program you created is one of the simplest VB.NET programs imaginable, but it is a complete VB.NET program, and it can be used to examine many of the elements common to VB.NET programs.

## Examining Your First Program

The single greatest challenge when learning to program is that you must learn everything before you can learn anything. Even this simple “Hello World” program uses many features of the language that will be discussed in coming chapters, including classes, namespaces, statements, static methods, objects, strings, inheritance, blocks, libraries, and even something called polymorphism!

It is as if you were learning to drive a car. You must learn to steer, accelerate, brake, and understand the flow of traffic. Right now we're going to get you out on the highway and just let you steer for a while. Over time you'll learn how to speed up and slow down. Along the way you'll learn to set the radio and adjust the heat so that you'll be more comfortable. In no time you'll be driving, and then won't your parents begin to worry.

### Line-by-Line Analysis

Hang on tight; we're going to zip through this quickly and come back to the details in subsequent chapters. The first line in the program defines a programming unit known as a *module*. In this case, the module is named HelloWorld:

```
Module HelloWorld
```

You begin each module definition using the `Module` keyword, as in the preceding code line. Likewise, you end each module definition with the line:

```
End Module
```

Within the module definition, you write other programming constructs. For instance, you might define what is called an *object*. An object is an individual instance of a thing. Every object belongs to a more general category known as a *class*. While a class defines a type, each instance of that type is an object (much as `Car` defines a type of vehicle, and your aging rust-bucket is an individual instance of `Car`).

In VB.NET there are thousands of classes. Classes are used to define Windows controls (buttons, list boxes, etc.), as well as types of things (employees, students, telephones, etc.) in the program you are writing. Some classes you create yourself; some you obtain from the .NET Framework. Each class must be named. Classes are the core of VB.NET and object-oriented programming.

For now, keep in mind that modules are actually related to classes. Technically, modules are a holdover from the previous generation of the VB language, VB6. In order to adapt VB6 for object-oriented programming, VB.NET converts modules to classes for you. You'll learn about classes in Chapter 3 and about modules and classes in Chapter 8.

Within the HelloWorld module, you define a *method* called Main(). A method is a small block of code that performs an action. The Main() method is the “entry point” for every VB.NET console application; it is where your program begins. Within the HelloWorld module, the Main() method is defined from lines 3 through 5. Notice the Sub keyword signals the beginning of the subroutine and the End Sub line concludes the method:

```
Sub Main()  
    System.Console.WriteLine("Hello world!")  
End Sub
```

Typically, one method calls another. The called method will do work, and it can return a value to the calling method. In VB.NET, methods come in two flavors: a method that returns a value is called a *function*; a method that does not return a value is called a *sub* (for subroutine). (Function and subroutine are old, non-object-oriented terms for these kinds of methods.) You'll see how methods call one another and return values in Chapter 9.

Main() is called by the operating system (when the program is invoked). Every method name is followed by opening and closing parentheses:

```
Sub Main()
```

As the parentheses imply, it is possible to pass values into a method so that the method can manipulate or use those values. These values are called *parameters* or *arguments* to the method. In this case, Main() has no arguments. (Method arguments are covered in Chapter 9.)

Within Main() is a single line of code:

```
System.Console.WriteLine("Hello world!")
```

WriteLine() is a method that is called by the Main() method; more about WriteLine() shortly. The Console is an object that represents your screen. In this case, each Console object belongs to the Console class. In VB.NET, classes can exist within a more comprehensive grouping known as a *namespace*.

In the HelloWorld program, the Console class is defined within the System namespace. The Console class has a method, WriteLine(), that displays a line of text to the screen. The complete identification for the WriteLine() method includes the class and namespace to which it belongs:

```
System.Console.WriteLine("Hello world!")
```

The WriteLine() method declares a single parameter, the text string you want to display. When you pass in a string to the method, the string is an argument. In our sample program, the string “Hello world!” corresponds to the parameter the method expects; thus, the string is displayed to the screen.

If you will be using many objects from the same namespace, you can save typing by telling the compiler that many of the objects you’ll be referring to are in that namespace. You do so by adding an Imports declaration to the beginning of your program:

```
Imports System
```

Once you add this line, you can use the Console class name without explicitly identifying its namespace (System). Thus, if you add the preceding Imports declaration, you can rewrite the contents of Main() as follows:

```
Console.WriteLine("Hello world!")
```

The compiler will check the namespace you identified (System), and it will find the Console class defined there.

Since the method (or sub) is defined within the module, you do not close the module until you have closed the method. Thus, the program ends with the sequence:

```
End Sub  
End Module
```

This discussion has omitted a single line in our program. Just before the start of the Main() method appears a comment (here in bold):

```
' every console app starts with Main  
Sub Main()  
    System.Console.WriteLine("Hello world!")
```

A comment is just a note to yourself. You insert comments to make the code more readable to programmers. You can place comments anywhere in your program that you think the explanation will be helpful; they have no effect on the running program.

In VB.NET, comments begin with a single quotation mark. The quote indicates that everything to the right on the same line is a comment and will be ignored by the VB.NET compiler.

Whew! That was a lot to take in all at once! Don’t panic; all of the concepts introduced here are explained in detail in later chapters.