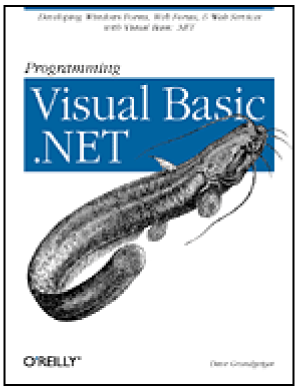


Programming Visual Basic .NET

Dave Grundgeiger
Publisher: O'Reilly
First Edition January 2002
ISBN: 0-596-00093-6, 464 pages

Published just in time for the first release of Visual Basic Studio .NET, *Programming Visual Basic .NET* is a programmer's complete guide to Visual Basic .NET. Starting with a sample application and a high-level map, the book jumps right into showing how the parts of .NET fit with Visual Basic .NET. Topics include the common language runtime Windows Forms, ASP.NET, Web Forms, Web Services, and ADO.NET.

Preface 9
Organization of This Book 9
Conventions Used in This Book 9
How to Contact Us 10
Acknowledgments 11
Chapter 1. Introduction 13
1.1 What Is the Microsoft .NET Framework? 13
1.2 What Is Visual Basic .NET? 14
1.3 An Example Visual Basic .NET Program 14
Chapter 2. The Visual Basic .NET Language 23
2.1 Source Files 23
2.2 Identifiers 23
2.3 Keywords 24
2.4 Literals 27
2.5 Types 31
2.6 Namespaces 40
2.7 Symbolic Constants 42
2.8 Variables 43
2.9 Scope 44
2.10 Access Modifiers 44
2.11 Assignment 45
2.12 Operators and Expressions 46
2.13 Statements 52
2.14 Classes 60
2.15 Interfaces 85
2.16 Structures 88
2.17 Enumerations 91
2.18 Exceptions 93
2.19 Delegates 98
2.20 Events 101
2.21 Standard Modules 104
2.22 Attributes 104
2.23 Conditional Compilation 108
2.24 Summary 109
Chapter 3. The .NET Framework 111
3.1 Common Language Infrastructure (CLI) and Common Language Runtime (CLR) 111
3.2 Common Type System (CTS) 111
3.3 Portions of the CLI 112
3.4 Modules and Assemblies 113
3.5 Application Domains 116
3.6 Common Language Specification (CLS) 116
3.7 Intermediate Language (IL) and Just-In-Time (JIT) Compilation 117
3.8 Metadata 117
3.9 Memory Management and Garbage Collection 118
3.10 A Brief Tour of the .NET Framework Namespaces 122

3.11 Configuration ... 125
3.12 Summary ... 131
Chapter 4. Windows Forms I: Developing Desktop Applications ... 133
4.1 Creating a Form ... 133
4.2 Handling Form Events ... 143
4.3 Relationships Between Forms ... 145
4.4 MDI Applications ... 147
4.5 Component Attributes ... 155
4.6 2-D Graphics Programming with GDI+ ... 160
4.7 Printing ... 174
4.8 Summary ... 186
Chapter 5. Windows Forms II: Controls, Common Dialog Boxes, and Menus ... 187
5.1 Common Controls and Components ... 187
5.2 Control Events ... 204
5.3 Form and Control Layout ... 204
5.4 Common Dialog Boxes ... 210
5.5 Menus ... 215
5.6 Creating a Control ... 227
5.7 Summary ... 236
Chapter 6. ASP.NET and Web Forms: Developing Browser-Based Applications 237
6.1 Creating a Web Form ... 238
6.2 Handling Page Events ... 251
6.3 More About Server Controls ... 253
6.4 Adding Validation ... 268
6.5 Using Directives to Modify Web Page Compilation ... 283
6.6 ASP.NET Objects: Interacting with the Framework ... 291
6.7 Discovering Browser Capabilities ... 296
6.8 Maintaining State ... 298
6.9 Application-Level Code and global.asax ... 304
6.10 Web-Application Security ... 307
6.11 Designing Custom Controls ... 320
6.12 Summary ... 328
Chapter 7. Web Services ... 329
7.1 Creating a Web Service ... 329
7.2 Testing a Web Service with a Browser ... 333
7.3 Web-Service Descriptions ... 335
7.4 Consuming a Web Service ... 335
7.5 Web-Service Discovery ... 340
7.6 Limitations of Web Services ... 340
7.7 Summary ... 341
Chapter 8. ADO.NET: Developing Database Applications ... 343
8.1 A Brief History of Universal Data Access ... 343
8.2 Managed Providers ... 343
8.3 Connecting to a SQL Server Database ... 344
SQL Server Authentication ... 347
8.4 Connecting to an OLE DB Data Source ... 348
8.5 Reading Data into a DataSet ... 349

8.6 Relations Between DataTables in a DataSet 360
8.7 The DataSet's XML Capabilities 362
8.8 Binding a DataSet to a Windows Forms DataGrid............................. 364
8.9 Binding a DataSet to a Web Forms DataGrid...................................... 367
8.10 Typed DataSets 368
8.11 Reading Data Using a DataReader... 370
8.12 Executing Stored ProceduresThrough a SqlCommand Object 371
8.13 Summary.. 374
Appendix A. Custom Attributes Defined in the System Namespace 375
Appendix B. Exceptions Defined in the System Namespace.................................. 381
Appendix D. Resources for Developers 391
D.1 .NET Information.. 391
D.2 Discussion Lists.. 392
Netiquette 392
Appendix E. Math Functions.. 395
Colophon... 398

Programming Visual Basic .NET

Preface
- Organization of This Book
- Conventions Used in This Book
- How to Contact Us
- Acknowledgments

1. Introduction
- 1.1 What Is the Microsoft .NET Framework?
- 1.2 What Is Visual Basic .NET?
- 1.3 An Example Visual Basic .NET Program

2. The Visual Basic .NET Language
- 2.1 Source Files
- 2.2 Identifiers
- 2.3 Keywords
- 2.4 Literals
- 2.5 Types
- 2.6 Namespaces
- 2.7 Symbolic Constants
- 2.8 Variables
- 2.9 Scope
- 2.10 Access Modifiers
- 2.11 Assignment
- 2.12 Operators and Expressions
- 2.13 Statements
- 2.14 Classes
- 2.15 Interfaces
- 2.16 Structures
- 2.17 Enumerations
- 2.18 Exceptions
- 2.19 Delegates
- 2.20 Events
- 2.21 Standard Modules
- 2.22 Attributes
- 2.23 Conditional Compilation
- 2.24 Summary

3. The .NET Framework
- 3.1 Common Language Infrastructure (CLI) and Common Language Runtime (CLR)
- 3.2 Common Type System (CTS)
- 3.3 Portions of the CLI
- 3.4 Modules and Assemblies
- 3.5 Application Domains
- 3.6 Common Language Specification (CLS)
- 3.7 Intermediate Language (IL) and Just-In-Time (JIT) Compilation
- 3.8 Metadata
- 3.9 Memory Management and Garbage Collection
- 3.10 A Brief Tour of the .NET Framework Namespaces
- 3.11 Configuration
- 3.12 Summary

4. Windows Forms I: Developing Desktop Applications
- 4.1 Creating a Form
- 4.2 Handling Form Events
- 4.3 Relationships Between Forms

4.4 MDI Applications
4.5 Component Attributes
4.6 2-D Graphics Programming with GDI+
4.7 Printing
4.8 Summary

5. Windows Forms II: Controls, Common Dialog Boxes, and Menus
5.1 Common Controls and Components
5.2 Control Events
5.3 Form and Control Layout
5.4 Common Dialog Boxes
5.5 Menus
5.6 Creating a Control
5.7 Summary

6. ASP.NET and Web Forms: Developing Browser-Based Applications
6.1 Creating a Web Form
6.2 Handling Page Events
6.3 More About Server Controls
6.4 Adding Validation
6.5 Using Directives to Modify Web Page Compilation
6.6 ASP.NET Objects: Interacting with the Framework
6.7 Discovering Browser Capabilities
6.8 Maintaining State
6.9 Application-Level Code and global.asax
6.10 Web-Application Security
6.11 Designing Custom Controls
6.12 Summary

7. Web Services
7.1 Creating a Web Service
7.2 Testing a Web Service with a Browser
7.3 Web-Service Descriptions
7.4 Consuming a Web Service
7.5 Web-Service Discovery
7.6 Limitations of Web Services
7.7 Summary

8. ADO.NET: Developing Database Applications
8.1 A Brief History of Universal Data Access
8.2 Managed Providers
8.3 Connecting to a SQL Server Database
8.4 Connecting to an OLE DB Data Source
8.5 Reading Data into a DataSet
8.6 Relations Between DataTables in a DataSet
8.7 The DataSet's XML Capabilities
8.8 Binding a DataSet to a Windows Forms DataGrid
8.9 Binding a DataSet to a Web Forms DataGrid
8.10 Typed DataSets
8.11 Reading Data Using a DataReader
8.12 Executing Stored ProceduresThrough a SqlCommand Object
8.13 Summary

A. Custom Attributes Defined in the System Namespace
AttributeUsage
CLSCompliant
ContextStatic

Flags
LoaderOptimization
MTAThread
NonSerialized
Obsolete
ParamArray
Serializable
STAThread
ThreadStatic

B. Exceptions Defined in the System Namespace

C. Cultures

D. Resources for Developers
D.1 .NET Information
D.2 Discussion Lists

E. Math Functions

Colophon

Preface

The purpose of this book is to provide experienced software developers with the means to quickly become productive in Microsoft's Visual Basic .NET development environment. The only assumption I make about you as a programmer is that you're comfortable with the concepts and processes of software development. This book will not teach you how to program. However, if you're currently a working Visual Basic, C++, or Java developer, this book will help you transfer your existing skills to this new environment.

Organization of This Book

This book contains eight chapters and four appendixes.

Chapter 1 starts out with three short *hello, world* examples that show how to enter and compile a console app, a GUI app, and a browser app. This gives the reader immediate gratification. The chapter also provides an overview of the .NET Framework and Visual Basic .NET.

Chapter 2 examines the syntax and use of the Visual Basic .NET language. This will not teach someone how to program, but it will teach a programmer how to program in Visual Basic .NET.

Chapter 3 explains the various components of the .NET Framework and explains why the .NET Framework is a Good Thing.

Chapter 4 explains how to use the Windows Forms class library for building GUI applications.

Chapter 5 picks up where Chapter 4 left off by discussing individual controls, showing how to use the common dialog boxes available in the .NET Framework, and examining menu creation and use.

Chapter 6 explains how to use the Web Forms class library for building browser-based applications.

Chapter 7 talks about building components that provide services over the Internet and how to consume those services.

Chapter 8 explains the distributed, stateless, disconnected data model encapsulated by ADO.NET.

Appendix A provides a list of the types known as attributes. The concept of attributes is discussed in Chapter 2.

Appendix B provides a list of system-generated exceptions. The concept of exceptions is discussed in Chapter 2.

Appendix C provides a list of culture names and IDs for globalization.

Appendix D provides a list of online resources where developers can get help and further information on Visual Basic .NET.

Appendix E lists the standard math functions that are available to the Visual Basic .NET programmer via the .NET Framework's Math class.

Conventions Used in This Book

Throughout this book, we've used the following typographic conventions:

Constant width

Constant width in body text indicates a language construct, such as the name of a stored procedure, a SQL statement, a Visual Basic .NET statement, an enumeration, an intrinsic or user-defined constant, a structure (i.e., a user-defined type), or an expression (like `dblElapTime = Timer - dblStartTime`). Code fragments and code examples appear exclusively in constant-width text. In syntax statements and prototypes, text set in constant width indicates such language elements as the function or procedure name and any invariable elements required by the syntax.

`Constant width italic`

Constant width italic in body text indicates parameter names. In syntax statements or prototypes, constant width italic indicates replaceable parameters. In addition, constant width italic is used in body text to denote variables.

Italic

Italicized words in the text indicate intrinsic or user-defined function and procedure names. Many system elements, such as paths and filenames, are also italicized. URLs and email addresses are italicized. Finally, italics are used for new terms where they are defined.

	This icon indicates a tip, suggestion, or general note.
	This icon indicates a warning or caution.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

There is a web page for this book, where we list errata, examples, or any additional information. You can access this page at:

http://www.oreilly.com/catalog/progvbdotnet

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Acknowledgments

Thank you to the folks at Microsoft who were willing to answer my incessant questions, even in the midst of having to meet their own delivery deadlines. This list of top-notch people includes Brad Abrams, Alan Carter, Kit George, Scott Guthrie, Jim Hogg, Rob Howard, and Susan Warren. Several of these people also read major portions of the manuscript and offered constructive comments.

Thank you to my coworkers at Tara Software, Inc., for letting me use them as sounding boards and for assisting with technical issues. This includes Dan Boardman, Kevin Caswick, Varon Fugman, Anson Goldade, Karl Hauth, Garrett Peterson, Dan Phelps, Scott Rassbach, and Adam Steinert.

Thank you to Tara Software, Inc., and particularly to its principals, Roger Mills, Lynne Pilsner, and Larry Kleopping, for supporting this project (emotionally and financially).

Thank you to O'Reilly & Associates, Inc. for letting me write the book that I felt needed to be written. Thanks in particular to my editor, Ron Petrusha, who always knows what to mess with and what to leave alone. Thanks also to Budi Kurniawan for graciously granting me permission to use material that he had written on Windows controls.

And finally, thank you to my friend and wife, Annemarie Newman. Annemarie, you've supported all my endeavors—from shareware with lots of downloads and zero payments to books that take longer to write than they should. Thank you. I think you should start filling out that graduate school application, angel. It's your turn.

Chapter 1. Introduction

With its release for the .NET platform, the Visual Basic language has undergone dramatic changes. For example:

- The language itself is now fully object-oriented.
- Applications and components written in Visual Basic .NET have full access to the .NET Framework, an extensive class library that provides system and application services.
- All applications developed using Visual Basic .NET run within a managed runtime environment, the .NET common language runtime.

In this introduction, I briefly discuss these changes and other changes before showing you three very simple, but complete, Visual Basic .NET applications.

1.1 What Is the Microsoft .NET Framework?

The *.NET Framework* encompasses the following:

- *A new way to expose operating system and other APIs.* For years, the set of Windows functionality that was available to developers and the way that functionality was invoked were dependent on the language environment being used. For example, the Windows operating system provides the ability to create windows (obviously). Yet, the way this feature was invoked from a C++ program was dramatically different from the way it was invoked from a Visual Basic program. With .NET, the way that operating system services are invoked is uniform across all languages (including code embedded in ASP.NET pages).

  This portion of .NET is commonly referred to as the *.NET Framework class library*.

- *A new infrastructure for managing application execution.* To provide a number of sophisticated new operating-system services—including code-level security, cross-language class inheritance, cross-language type compatibility, and hardware and operating-system independence, among others—Microsoft developed a new runtime environment known as the Common Language Runtime (CLR). The CLR includes the Common Type System (CTS) for cross-language type compatibility and the Common Language Specification (CLS) for ensuring that third-party libraries can be used from all .NET-enabled languages.

  To support hardware and operating-system independence, Microsoft developed the Microsoft Intermediate Language (MSIL, or just IL). IL is a CPU-independent machine language-style instruction set into which .NET Framework programs are compiled. IL programs are compiled to the actual machine language on the target platform prior to execution (known as *just-in-time*, or JIT, compiling). IL is never interpreted.

- *A new web server paradigm.* To support high-capacity web sites, Microsoft has replaced its Active Server Pages (ASP) technology with ASP.NET. While developers who are used to classic ASP will find ASP.NET familiar on the surface, the underlying engine is different, and far more features are supported. One difference, already mentioned in this chapter, is that ASP.NET web page code is now compiled rather than interpreted, greatly increasing execution speed.
- *A new focus on distributed-application architecture.*Visual Studio .NET provides top-notch tools for creating and consuming *web services* -- vendor-independent software services that can be invoked over the Internet.

  The .NET Framework is designed top to bottom with the Internet in mind. For example, ADO.NET, the next step in the evolution of Microsoft's vision of "universal data access," assumes that applications will work with disconnected data by default. In addition, the

ADO.NET classes provide sophisticated XML capabilities, further increasing their usefulness in a distributed environment.

An understanding of the .NET Framework is essential to developing professional Visual Basic .NET applications. The .NET Framework is explained in detail in Chapter 3.

1.2 What Is Visual Basic .NET?

Visual Basic .NET is the next generation of Visual Basic, but it is also a significant departure from previous generations. Experienced Visual Basic 6 developers will feel comfortable with Visual Basic .NET code and will recognize most of its constructs. However, Microsoft has made some changes to make Visual Basic .NET a better language and an equal player in the .NET world. These include such additions as a `Class` keyword for defining classes and an `Inherits` keyword for object inheritance, among others. Visual Basic 6 code can't be compiled by the Visual Basic .NET compiler without significant modification. The good news is that Microsoft has provided a migration tool to handle the task (mostly, anyway). Code migration is explained in Appendix A. The Visual Basic .NET language itself is detailed in Chapter 2.

Over the last several months I have spent almost all of my time playing with .NET and writing Visual Basic .NET programs. As a user of Visual Basic since Version 4, I can tell you that I am pleased with this new technology and with the changes that have been made to Visual Basic. In my opinion, Microsoft has done it right.

1.3 An Example Visual Basic .NET Program

The first program to write is the same for all languages: Print the words hello, world

—Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*

It has become a tradition for programming books to begin with a *hello, world* example. The idea is that entering and running a program—any program—may be the biggest hurdle faced by experienced programmers approaching a new platform or language. Without overcoming this hurdle, nothing else can follow. This chapter contains three such examples: one that creates a console application, one that creates a GUI application, and one that creates a browser-based application. Each example stands alone and can be run as is. The console and GUI applications can both be compiled from the command line (yes, Visual Basic .NET has a command-line compiler!). The browser-based application requires a computer running Internet Information Server (IIS).

1.3.1 hello, world

This is the world's favorite programming example, translated to Visual Basic .NET:

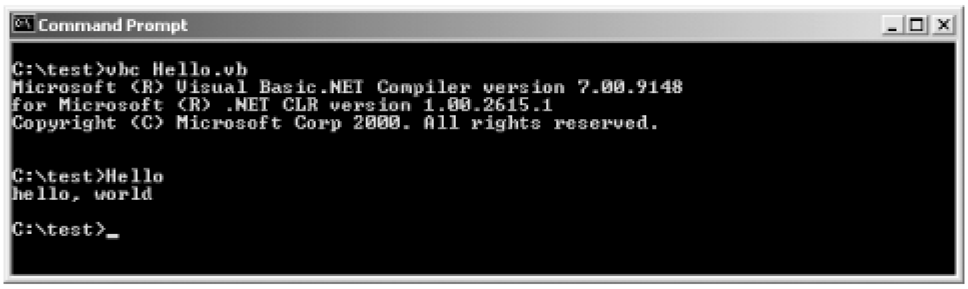
```
Imports System

Public Module Hello
   Public Sub Main(  )
      Console.WriteLine("hello, world")
   End Sub
End Module
```

This version of *hello, world* is a *console application* -- it displays its output in a Windows command-prompt window. To compile this program, enter it using any text editor, such as Windows's Notepad, save it in a file whose name ends with *.vb*, such as *Hello.vb*, and compile it from the Windows command line with this command:

```
vbc Hello.vb
```

The command `vbc` invokes the Visual Basic .NET command-line compiler, which ships with the .NET Framework SDK, and instructs it to compile the file named in the command-line argument. Compiling *Hello.vb* generates the file *Hello.exe*. After compiling, type `Hello` at the command line to run your program. Figure 1-1 shows the results of compiling and running this program.

Figure 1-1. Compiling and running hello, world


Command Prompt

```
C:\test>vbc Hello.vb
Microsoft (R) Visual Basic.NET Compiler version 7.00.9148
for Microsoft (R) .NET CLR version 1.00.2615.1
Copyright (C) Microsoft Corp 2000. All rights reserved.


C:\test>Hello
hello, world

C:\test>_
```

If you're accustomed to programming in Visual Basic 6, you can see even from this little program that Visual Basic has changed dramatically. Here's a breakdown of what's happening in this code.

The first line:

```
Imports System
```

indicates that the program may use one or more types defined in the System *namespace*. (Types are grouped into namespaces to help avoid name collisions and to group related types together.) Specifically, the *hello, world* program uses the Console class, which is defined in the System namespace. The `Imports` statement is merely a convenience. It is not needed if the developer is willing to qualify type names with their namespace names. For example, the *hello, world* program could have been written this way:

```
Public Module Hello
   Public Sub Main(  )
      System.Console.WriteLine("hello, world")
   End Sub
End Module
```

However, it is customary to use the `Imports` statement to reduce keystrokes and visual clutter.

An important namespace for Visual Basic developers is Microsoft.VisualBasic. The types in this namespace expose members that form Visual Basic's intrinsic functions and subroutines. For example, the Visual Basic *Trim* function is a member of the Microsoft.VisualBasic.Strings class, while the *MsgBox* function is a member of the Microsoft.VisualBasic.Interaction class. In addition, Visual Basic's intrinsic constants come from enumerations within this namespace. Much of the functionality available in this namespace, however, is also duplicated within the .NET Framework's Base Class Library. Developers who are not familiar with Visual Basic 6 will likely choose to ignore this namespace, favoring the functionality provided by the .NET Framework. The .NET Framework is introduced later in this chapter and is explained in detail in Chapter 3.

Next, consider this line:

```
Public Module Hello
```

This line begins the declaration of a standard module named `Hello`. The standard-module declaration ends with this line:

```
End Module
```

In Visual Basic 6, various program objects were defined by placing source code in files having various filename extensions. For example, code that defined classes was placed in *.cls* files, code that defined standard modules was placed in *.bas* files, and so on. In Visual Basic .NET, all source files have *.vb* filename extensions, and program objects are defined with explicit syntax. For example, classes are defined with the `Class...End Class` construct, and standard modules are defined with the `Module...End Module` construct. Any particular *.vb* file can contain as many of these declarations as desired.

The purpose of standard modules in Visual Basic 6 was to hold code that was outside of any class definition. For example, global constants, global variables, and procedure libraries were often placed in standard modules. Standard modules in Visual Basic .NET serve a similar purpose and can be used in much the same way. However, in Visual Basic .NET they define datatypes that cannot be instantiated and whose members are all static. This will be discussed in more detail in Chapter 2.

The next line in the example begins the definition of a subroutine named *Main*:

```
Public Sub Main(  )
```

It ends with:

```
End Sub
```

This syntax is similar to Visual Basic 6. The `Sub` statement begins the definition of a *subroutine* -- a method that has no return value.

The *Main* subroutine is the entry point for the application. When the Visual Basic .NET compiler is invoked, it looks for a subroutine named *Main* in one of the classes or standard modules exposed by the application. If *Main* is declared in a class rather than in a standard module, the subroutine must be declared with the `Shared` modifier. This modifier indicates that the class does not need to be instantiated for the subroutine to be invoked. In either case, the *Main* subroutine must be `Public`. An example of enclosing the *Main* subroutine in a class rather than in a standard module is given at the end of this section.

If no *Main* subroutine is found, or if more than one is found, a compiler error is generated. The command-line compiler has a switch (`/main:`*`location`*) that allows you to specify which class or standard module contains the *Main* subroutine that is to be used, in the case that there is more than one.

Lastly, there's the line that does the work:

```
Console.WriteLine("hello, world")
```

This code invokes the Console class's WriteLine method, which outputs the argument to the console. The WriteLine method is defined as a *shared* (also known as a *static*) method. Shared methods don't require an object instance in order to be invoked; nonshared methods do. Shared methods are invoked by qualifying them with their class name (in this case, Console).

Here is a program that uses a class instead of a standard module to house its *Main* subroutine. Note that *Main* is declared with the `Shared` modifier. It is compiled and run in the same way as the standard module example, and it produces the same output. There is no technical reason to choose one implementation over the other.

```
Imports System

Public Class Hello
   Public Shared Sub Main(  )
      Console.WriteLine("hello, world")
   End Sub
End Class
```

1.3.2 Hello, Windows

Here's the GUI version of *hello, world*:

```
Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class HelloWindows

   Inherits Form

   Private lblHelloWindows As Label

   Public Shared Sub Main(  )
      Application.Run(New HelloWindows(  ))
   End Sub

   Public Sub New(  )

      lblHelloWindows = New Label(  )
      With lblHelloWindows
         .Location = New Point(37, 31)
         .Size = New Size(392, 64)
         .Font = New Font("Arial", 36)
         .Text = "Hello, Windows!"
         .TabIndex = 0
         .TextAlign = ContentAlignment.TopCenter
      End With

      Me.Text = "Programming Visual Basic .NET"
      AutoScaleBaseSize = New Size(5, 13)
      FormBorderStyle = FormBorderStyle.FixedSingle
      ClientSize = New Size(466, 127)

      Controls.Add(lblHelloWindows)

   End Sub

End Class
```

This is similar to the *hello, world* console application, but with extra stuff required since this is a GUI application. Two additional `Imports` statements are needed for drawing the application's window:

```
Imports System.Drawing
Imports System.Windows.Forms
```

The HelloWindows class has something that Visual Basic programs have never seen before, the `Inherits` statement:

```
Inherits Form
```

The Visual Basic .NET language has class inheritance. The HelloWindows class inherits from the Form class, which is defined in the System.Windows.Forms namespace. Class inheritance and the `Inherits` statement are discussed in Chapter 2.

The next line declares a label control that will be used for displaying the text `Hello, Windows`:

```
Private lblHelloWindows As Label
```

The Label class is defined in the System.Windows.Forms namespace.

As is the case with console applications, GUI applications must have a shared subroutine called *Main*:

```
Public Shared Sub Main(  )
   Application.Run(New HelloWindows(  ))
End Sub
```

This Main method creates an instance of the HelloWindows class and passes it to the Run method of the Application class (defined in the System.Windows.Forms namespace). The Run method takes care of the housekeeping of setting up a Windows *message loop* and hooking the HelloWindows form into it.

Next is another special method:

```
Public Sub New(  )
```

Like *Main*, *New* has special meaning to the Visual Basic .NET compiler. Subroutines named *New* are compiled into *constructors*. A constructor is a method that has no return value (but can have arguments) and is automatically called whenever a new object of the given type is instantiated. Constructors are explained further in Chapter 2.

The constructor in the HelloWindows class instantiates a Label object, sets some of its properties, sets some properties of the form, and then adds the Label object to the form's Controls collection. The interesting thing to note is how different this is from how Visual Basic 6 represented form design. In Visual Basic 6, form layout was represented by data in *.frm* files. This data was not code, but rather a listing of the properties and values of the various elements on the form. In Visual Basic .NET, this approach is gone. Instead, Visual Basic .NET statements must explicitly instantiate visual objects and set their properties. When forms are designed in Visual Studio .NET using its drag-and-drop designer, Visual Studio .NET creates this code on your behalf.

The command line to compile the *Hello, Windows* program is:

```
vbc HelloWindows.vb
/reference:System.dll,System.Drawing.dll,System.Windows.Forms.dll
/target:winexe
```

(Note that there is no break in this line.)

The command line for compiling the *Hello, Windows* program has more stuff in it than the one for the console-based *hello, world* program. In addition to specifying the name of the *.vb* file, this command line uses the `/references` switch to specify three *.dll*s that contain the implementations of library classes used in the program (Form, Label, Point, etc.). The *hello, world* console application didn't require references when being compiled because all it used was the Console class, defined in the System namespace. The Visual Basic .NET command-line compiler includes two references implicitly: *mscorlib.dll* (which contains the System namespace) and *Microsoft.VisualBasic.dll* (which contains helper classes used for implementing some of the features of Visual Basic .NET).

Besides the `/references` switch, the command line for compiling the *Hello, Windows* program includes the `/target` switch. The `/target` switch controls what kind of executable code file is produced. The possible values of the `/target` switch are:

exe

Creates a console application. The generated file has an extension of *.exe*. This is the default.

winexe

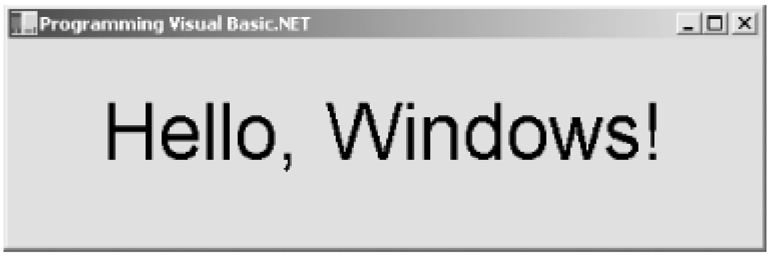
Creates a GUI application. The generated file has an extension of *.exe*.

library

Creates a class library. The generated file has an extension of *.dll*.

The output of *Hello, Windows* is shown in Figure 1-2.

Figure 1-2. Hello, Windows!



GUI applications are explained in detail in Chapter 4 and Chapter 5.

1.3.3 Hello, Browser

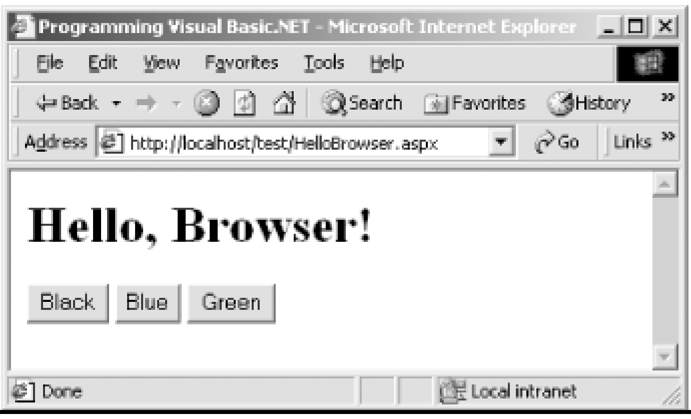
Here is a browser-based version of the *hello, world* application. Because the simplest version of such an application could be accomplished with only HTML, I've added a little spice. This web page includes three buttons that allow the end user to change the color of the text.

```
<script language="VB" runat="server">

      Sub Page_Load(Sender As Object, E As EventArgs)
         lblMsg.Text = "Hello, Browser!"
      End Sub

      Sub btnBlack_Click(Sender As Object, E As EventArgs)
         lblMsg.ForeColor = System.Drawing.Color.Black
      End Sub

      Sub btnGreen_Click(Sender As Object, E As EventArgs)
         lblMsg.ForeColor = System.Drawing.Color.Green
      End Sub

      Sub btnBlue_Click(Sender As Object, E As EventArgs)
         lblMsg.ForeColor = System.Drawing.Color.Blue
      End Sub

</script>

<html>

   <head>
      <title>Programming Visual Basic .NET</title>
   </head>

   <body>
```

```
      <form action="HelloBrowser.aspx" method="post" runat="server">
         <h1><asp:label id="lblMsg" runat="server"/></h1>
         <p>
            <asp:button type="submit" id="btnBlack" text="Black"
               OnClick="btnBlack_Click" runat="server"/>
            <asp:button id="btnBlue" text="Blue"
               OnClick="btnBlue_Click" runat="server"/>
            <asp:button id="btnGreen" text="Green"
               OnClick="btnGreen_Click" runat="server"/>
         </p>
      </form>
   </body>

</html>
```

To run this program, enter it using a text editor and save it in a file named *HelloBrowser.aspx*. Because the application is a web page that is meant to be delivered by a web server, it must be saved onto a machine that is running IIS and has the .NET Framework installed. Set up a virtual folder in IIS to point to the folder containing *HelloBrowser.aspx*. Finally, point a web browser to *HelloBrowser.aspx*. The output of the *Hello, Browser* application is shown in Figure 1-3.

Figure 1-3. Hello, Browser!


Programming Visual Basic.NET - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost/test/HelloBrowser.aspx Go Links
Hello, Browser!
Black Blue Green
Done Local intranet

Be sure to reference the file through the web server machine name or *localhost* (if the web server is on your local machine), so that the web server is invoked. For example, if the file is in a virtual directory called *Test* on your local machine, point your browser to *http://localhost/Test/HelloBrowser.aspx*. If you point your browser directly to the file using a filesystem path, the web server will not be invoked.

Going into detail on the *Hello, Browser* code would be too much for an introduction. However, I'd like to draw your attention to the `<asp:label>` and `<asp:button>` tags. These tags represent *server-side controls*. A server-side control is a class that is instantiated on the web server and generates appropriate output to represent itself on the browser. These classes have rich, consistent sets of properties and methods and can be referenced in code like controls on forms are referenced in GUI applications.

ASP.NET has many other nifty features, some of which are:

- Web pages are compiled, resulting in far better performance over classic ASP.
- Code can be pulled out of web pages entirely and placed in *.vb* files (called *code-behind files*) that are referenced by the web pages. This separation of web page layout from code results in pages that are easier to develop and maintain.