Beginning Visual Web Programming in VB .NET

From Novice to Professional

DANIEL CAZZULINO
VICTOR GARCIA APREA
JAMES GREENWOOD
CHRIS HART

Apress®

Beginning Visual Web Programming in VB .NET: From Novice to Professional

ISBN (pbk): 1-59059-359-6

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Ewan Buckingham
Technical Reviewer: Victor Garcia Aprea
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Chris Mills, Dominic Shakeshaft, Jim Sumser
Assistant Publisher: Grace Wong
Project Manager: Beckie Stones
Copy Manager: Nicole LeClerc
Copy Editor: Marilyn Smith
Production Manager: Kari Brooks-Copony
Production Editor: Kelly Winquist
Compositor: Dina Quan
Proofreader: Katie Stence
Indexer: Kevin Broccoli
Artist: Kinetic Publishing
Cover Designer: Kurt Krames
Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013, and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders@springer-ny.com`, or visit `http://www.springer-ny.com`. Outside the United States: fax +49 6221 345229, e-mail `orders@springer.de`, or visit `http://www.springer.de`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

To Any, for believing in me, changing for me, and risking with me on this seemingly never-ending journey though new challenges. I hope I can get to be as supportive with you as you have been with me. You have been the most amazing companion I could have ever dreamed of.

*To Agustina, my little baby, for giving me yet another reason to love working at home. Watching you grow every day up close is a great gift from God (I hear Any say I shouldn't be *just* watching :o)).*

To my father, for being such a life warrior. I'll never give up, just as you never do. Thanks for staying with us and giving me the opportunity of making you feel proud of me. I can clearly see now that it wasn't your time, and God really wanted you to see everything that came after.

To all my girls, mum and sisters, for showing me what great women look like. I must listen to you more. I'm working on that.

To my dear friends (you know who you are), for always being there and bringing me back to earth. Talking to you makes me understand there's a beautiful life outside (along with?) .NET.

To Olga, for being like a second mother, and to Pedro, for doing all the things I should do at home so I can spend more time programming, without getting Any mad at me. I never thank you enough. With regards to Pedro, it's already too late. I hope I never make that mistake again. We miss you a lot.

Daniel Cazzulino

Contents at a Glance

About the Authors xv
About the Technical Reviewer xvii
Acknowledgments xix
Introduction xxi

CHAPTER 1 Environment and Architecture 1
CHAPTER 2 Web Development in .NET 29
CHAPTER 3 User Interfaces and Server Controls 59
CHAPTER 4 ADO.NET 117
CHAPTER 5 Data Binding 155
CHAPTER 6 ASP.NET State Management 207
CHAPTER 7 Markup Languages and XML 259
CHAPTER 8 XML and Web Development 293
CHAPTER 9 Web Services in Web Applications 343
CHAPTER 10 ASP.NET Authentication, Authorization, and Security 389
CHAPTER 11 Debugging and Exception Handling 421
CHAPTER 12 Caching and Performance Tuning 475
CHAPTER 13 Publishing Web Applications in .NET 521
APPENDIX A The Friends Reunion Application 567
APPENDIX B Management of IIS and MSDE 573

INDEX 595

Contents

About the Authors xv
About the Technical Reviewer xvii
Acknowledgments xix
Introduction xxi

CHAPTER 1 Environment and Architecture 1

Web Application Considerations 1
The Web Model 2
Desktop Applications vs. Web Applications 3
Web Servers and Web Clients 4
System Configuration for Web Development 7
Installing and Configuring IIS for .NET Web Applications 8
Administering IIS 9
Configuring Virtual Directories 16
Dynamic Web Applications 17
Client-Side Processing and Server-Side Processing 18
An Introduction to State Management 24
Web Application Architecture 24
Summary 27

CHAPTER 2 Web Development in .NET 29

An Introduction to ASP.NET 29
Web Form Construction in VS .NET 30
Understanding Web Form Structure:
Presentation and Processing 31
Using the Properties Browser 40
Using Code-Behind 45
A Brief Tour of An ASP.NET Application 49
ASP.NET Application Files 49
The Class View 50
Object Orientation in ASP.NET 51
Reusability and Encapsulation 52

Compilation 53
The Lifecycle of a Page 54
Summary 56

CHAPTER 3 User Interfaces and Server Controls 59

Server Controls 60
HTML Controls 61
HTML Server Controls 72
Web Server Controls 76
Validation Controls 87
User Controls 94
Custom Controls 102
Dynamic Content 110
Avoiding Code Duplication 113
Summary 116

CHAPTER 4 ADO.NET 117

The Architecture of ADO.NET 117
The Data Reader Object 118
Data Providers 119
Programmatic Use of ADO.NET 121
Adding Data to a Database 121
Retrieving Data from a Database 126
Changing the Data in a Database 133
The Rest of the Picture: The DataSet and Data Adapter Objects 141
Dealing with Disconnected Data 141
Using a DataSet Object 143
Summary 154

CHAPTER 5 Data Binding 155

Introduction to Components 155
Placing a Component on a Form 156
Configuring Dynamic Properties 158
Data Binding 160
Using Binding Expressions 161
Formatting with the DataBinder Class 162
Using Data Binding 162
Binding to Sets of Data 170

Working Visually with Data 177
Working with Data Components 178
Using Typed Datasets 179
Advanced Data Binding 186
Paging 189
Freestyle Data Binding and Editing—The DataList 192
Summary 206

CHAPTER 6 ASP.NET State Management 207

State Storage and Scope 207
Session State 208
Controlling the Session State 220
Configuring the Session State 224
Application State 226
Using Application Object and Events 232
Viewstate 233
Using the Viewstate As a Datastore 237
Transient State 243
Cookies 248
Passing Data with Query Strings 255
Passing Data with Hidden Form Fields 256
Summary 256

CHAPTER 7 Markup Languages and XML 259

Markup Languages 260
HyperText Markup Language 261
Extensible Markup Language (XML) 262
XML Data Exchange 267
XML Schemas and Validation 268
Markup Languages, Schemas, and Validation 270
XML Schema Creation 272
Using XML Namespaces 274
Building an XML Schema 276
Defining Complex Types 280
Defining Custom Simple Types 286
Restricting Element Occurrence 288
Viewing the Entire Schema 289
Summary 290

CHAPTER 8 XML and Web Development 293

XML Document Creation in VS .NET 294
Creating XML Documents Visually 294
Creating XML Documents in the Data View 298
Programmatic Manipulation of XML in .NET 300
Reading and Validating XML 300
Receiving the Uploaded File 308
XML Queries with XPath 321
Querying Document Object Model (DOM) Documents 324
Understanding the XPath Data Model 329
Building XPath Expressions Dynamically 334
XML APIs Comparison 338
Summary 340

CHAPTER 9 Web Services in Web Applications 343

Overview of Web Services 343
Web Services Relationship to the Browser/Server Model 344
VS .NET Support for Web Services 345
Web Service Implementation 346
Implementing Web Methods 346
Testing the Web Service 349
Using Complex Data Types 352
Web Service Consumption 357
An Introduction to SOAP 363
Understanding the SOAP Message Format 363
Viewing a SOAP Request and Response 365
Error Handling in Web Services 366
Web Service Efficiency 370
Caching in Web Services 370
Reducing the Amount of Data Involved 371
Using Custom Data Types for Optimization 376
Adding State to Web Services 385
Third-Party Web Services 386
Summary 387

CHAPTER 10 ASP.NET Authentication, Authorization, and Security 389

Security Overview 389
Security Architecture 390

Essential Terminology 390
The ASP.NET Security Infrastructure 395
Application Security Configuration 398
Authentication Configuration 398
Authorization Configuration 399
Location Configuration 400
Authentication Modes 401
Windows Authentication 401
Passport Authentication 403
Forms Authentication 403
Customized Authentication and Role-Based Security 409
Implementing Custom Authentication 410
Securing Folders 416
Summary 419

CHAPTER 11 Debugging and Exception Handling 421

Types of Errors 422
Syntax Errors 422
Semantic Errors 424
Input Errors 426
Debugging Web Applications 426
ASP.NET Tracing 427
Tracing and Assertions in .NET 437
The VS .NET Debugger 442
Exceptions 459
Exception Handling 460
Unhandled Exceptions 469
Summary 474

CHAPTER 12 Caching and Performance Tuning 475

What Is Good Performance? 475
Performance Monitoring 476
Configuring the System Monitor in PerfMon 477
Avoiding External Overhead 484
Performance Testing an Application 485
Installing the Web Application Stress Tool 485
Generating a Realistic Set of Data 486
Preparing a Performance Test with a Simulated Load 487
Running the Performance Test 492

Caching 494
Caching Overview 495
ASP.NET Caching 498
Monitoring the Cache API 512
Controlling the Viewstate 512
Disabling the Viewstate for Controls 512
Disabling the Viewstate at the Page and Application Levels 514
Checking the Viewstate Encryption Features 515
Deciding What to Put in Viewstate 515
More ASP.NET Performance Tips 515
Server-Side Redirection Using Server.Transfer 515
Using Web Controls Conservatively 517
Disabling Session State 518
Improving Database Access 518
Summary 519

CHAPTER 13 Publishing Web Applications in .NET 521

Methods for Deploying .NET Applications 521
XCOPY Deployment 522
Deployment Projects 522
Manual Web Application Deployment 523
Setup Projects in VS .NET 528
Creating Web Setup Projects 530
Including the dotnetfx.exe File with Your Installation 536
Viewing Application Dependencies and Outputs 536
Using the Setup Project 538
Uninstalling a Project 539
Customized Deployment 540
Adding a Custom File 540
Editing the User Interface 541
Building the Project 546
Adding Custom Actions 548
Using Installer Classes 550
Deploying Application Configuration Settings 559
Using Launch Conditions 560
Summary 563
Web Applications—An Overview 564

APPENDIX A The Friends Reunion Application 567

Friends Reunion Database Design 567
How to Set Up the Code Download Package 568
Setting Up the Database 568
Setting Up the Code Samples 569
How to Create GUIDs for Database Keys 571

APPENDIX B Management of IIS and MSDE 573

IIS Configuration 573
Configuring Site-Wide Settings 573
Configuring ASP.NET Applications in IIS 578
Locking Down IIS 583
Impersonation Configuration 583
An Introduction to IIS 6 588
MSDE Management 590
Using MSDE 590
Administering Data via the Server Explorer 590

INDEX 595

About the Authors

DANIEL CAZZULINO, one of the original authors of this book, is the main editor of this edition. He reviewed and modified the original text to make it an even more consistent book and to adapt it to the latest changes in the ASP.NET product.

Daniel (also known as "kzu") is cofounder of Clarius Consulting (`http://clariusconsulting.net`), providing training, consulting, and development in Microsoft .NET technologies. He coauthored several books for Wrox Press and Apress on web development and server controls with ASP.NET, has written and reviewed many articles for *ASP Today* and *C# Today*, and currently enjoys sharing his .NET, XML, and ASP.NET experiences through his weblog (`http://clariusconsulting.net/kzu`). He also works closely with Microsoft in key projects from the Patterns and Practices group, shaping the use of OO techniques, design patterns, and frameworks for .NET. Microsoft recognized him as a Most Valuable Professional (MVP) on XML Technologies for his contributions to the community, mainly through the XML-savvy open source project NMatrix (`http://sourceforge.net/projects/dotnetopensrc`) he cofounded and his weblog. He also started the promising Mvp.Xml project with fellow XML MVP experts worldwide (`http://mvp-xml.sf.net`). Surprisingly enough, Daniel is a lawyer who found a more exciting career as a developer and .NET/XML geek.

JAMES GREENWOOD is a technical architect and author based in West Yorkshire, England. He spends his days (and most of his nights) designing and implementing .NET solutions—from government knowledge-management systems to mobile integration platforms—all the while waxing lyrical on the latest Microsoft technologies. His professional interests include research into distributed interfaces, the automation of application development, and human-machine convergence. When he can be pried away from the keyboard, James can be found out and about, indulging in his other great loves: British sports cards and Egyptology. You can reach James at `jsg@altervisitor.com`.

CHRIS HART is a full-time developer, part-time author based in Birmingham (UK). Chris took the long route into development, having originally studied Mechanical Engineering. She is currently working at Trinity Expert Systems (`www.tesl.com`), and her most recent project has involved developing a heavily customized Microsoft CMS Server application. In her spare time, she enjoys writing about ASP.NET, and she has contributed chapters to many books, published by both Apress and Wrox Press. In the remaining hours of the day, she enjoys cooking, going to the gym, and relaxing with her extremely supportive husband, James.

VICTOR GARCIA APREA, one of the authors of the original version of this book, also served as the technical reviewer of this edition. You can read about him on the "About the Technical Reviewer" page.

About the Technical Reviewer

VICTOR GARCIA APREA is cofounder of Clarius Consulting, providing training, consulting, and development in Microsoft .NET technologies. Victor has been involved with ASP.NET since early in its development and was recognized as a Microsoft MVP for ASP.NET since 2002. He has written books and articles, and also has done a lot of reviewing for Wrox Press, Apress, and Microsoft Press. Victor is a regular speaker at Microsoft Argentina (MSDN DevDays, Ask the Experts panel, and other events) and .NET local user groups.

You can read Victor's weblog at `http://clariusconsulting.net/vga`.

Acknowledgments

Special thanks to Chris Hart for sharing her writing skills with me and turning a chaotic, spasmodic writer into a bearable author. To Ian Nutt for letting me learn the intricacies of the editing work, which ultimately made my role in this book possible. The editorial industry misses you!

Thanks to Dominique, for remembering my name after the lunch, our meeting, and the years, and for believing in the strong and concise material we created for this book. Hey, I slipped Mono release only by a year :o)!

And thanks to Ewan, Marilyn, Beckie, and Kelly for their help in getting this book out the door.

And thanks to Victor for being the most amazing technical reviewer (besides being an excellent author and partner, too). It's great to see that just as we began, we continue (not that you being the reviewer and me the editor means I'm sort of your boss…).

—Daniel

Introduction

The introduction of .NET has blurred the lines between previously distinct programming disciplines, and it has done so to great effect for developers. With so much functionality encapsulated by the .NET Framework class library, some diverse tasks have gained a common programming interface.

One area in which this change is particularly striking is that of web development. Before .NET, web application programming "the Microsoft way" was all about ASP. At the time, ASP was new, accessible, and exciting. But it was also script-based and inefficient. It led to serious maintainability problems, and the IDEs were disjointed. Microsoft has channeled the lessons it learned from ASP into its .NET Framework. Now, with ASP.NET (the .NET web development technology), you can create efficient, interactive Internet applications using the same languages that you use for Windows desktop applications.

In Visual Studio .NET (VS .NET), Microsoft has taken this idea even further; not only does the *code* look similar, but the *GUI* looks similar, too. Visual Basic .NET's (VB. NET's) familiar form-based interface is used for development of web applications, as well as for desktop programs. If you want to, you can create a web application without ever seeing a line of HTML code, and you can take advantage of all the facilities for testing and debugging that VS .NET provides to programmers of all disciplines.

The structure of class libraries in the .NET Framework is such that the methodology you use is the same, regardless of whether you're developing desktop applications or web applications. ASP.NET is really just a series of classes in the .NET Framework, just like the Windows Forms classes. From this perspective, the move from desktop application development to web development shouldn't be too much of a leap.

Yet there *are* some major differences that you need to consider when you move to web development. We are no longer talking about applications installed and running on individual machines; instead, we're talking about hosting an application on a central server, ready to be accessed by hundreds or thousands of remote clients every hour. This means that you need to be more concerned with performance, scalability, and security issues to ensure that the end user's experience is as smooth and secure as possible.

In this book, we'll explain the issues involved in the web development paradigm and how they're different from those you're used to in desktop application development. To demonstrate how to apply these principles, beginning in Chapter 3, this book guides you through building a feature-rich, interactive web application called Friends Reunion, using VB .NET, ASP.NET, and VS .NET.

The emphasis is on learning by practice. Every example in the book is described step by step, and we'll outline and explain each stage in the development, debugging, and deployment of the Friends Reunion application.

Who Is This Book For?

This book is predominantly targeted at developers who have some experience in the VB .NET language (perhaps through practical application of the language or simply from a VB .NET tutorial book). These developers may fall into one of two groups:

- Readers who have little or no web development experience, have gained their VB .NET experience mostly in the context of desktop applications, and are seeking to apply this VB .NET expertise in web development in .NET
- Readers who have gained some web development experience using ASP, PHP, or other technologies, and are seeking to move into web development using .NET and VB .NET.

This book does not assume that you have programmed for the web environment before, but it does assume that you have some familiarity with VB .NET. Previous experience with the VS .NET integrated development environment (IDE) is not essential, but it is helpful. Similarly, we assume no previous experience with HTML, XML, databases, or any of the other technologies mentioned in this book—though a little background knowledge does no harm.

What Does This Book Cover?

The first two chapters of this book are introductory. They provide the basic foundation you need to begin working on the Friends Reunion web application in Chapter 3.

The remaining chapters examine different aspects of web application development using VB .NET and ASP.NET. In each chapter, we study an aspect both in general terms and within the context of the Friends Reunion application. Over the course of these 11 chapters, we build, test, debug, and deploy a rich interactive web application—and you'll see every single step. You can obtain all of the code presented in this book from the Downloads section of the Apress web site (`http://www.apress.com`). Appendix A contains an overview of the code and explains how to install and run it for each chapter.

Here's a summary of the contents of this book:

- Chapter 1 is an introduction to the web environment. It discusses the similarities and differences between web applications and desktop applications, and explains what happens behind the scenes when a user employs a browser to request a web page. The intention is to clarify the issues that influence the way we design applications for the Web and to set the scene for the remainder of the book. We also set up the web server here and create a couple of simple examples to get things started.
- In Chapter 2, we move on to create some basic ASP.NET web forms using VS .NET. We look at how web forms are processed and the lifecycle of a page, and we demonstrate it all by walking through our first ASP.NET application.
- Chapter 3 is all about the server control. The server control is the core part of any web form; it's at the heart of the development of dynamic, interactive web sites in .NET. VS .NET allows you to drag-and-drop server controls onto your web forms in exactly the same way that you insert Windows Forms controls into a Windows desktop application. And you can add code to your forms to interact with these controls in much the same way, too.

- In Chapters 4 and 5, we turn our attention to data. Most interactive web applications rely on the existence and manipulation of data of some form, and on storage of that data (either in a full-scale database or some other format). In Chapter 4, we use ADO.NET to access and manipulate data. In Chapter 5, we demonstrate how data binding techniques make it easy to display data on your pages in a user-friendly manner. You will also see how to apply templates to your web forms to alter the look and feel of your data-bound controls.

- Chapter 6 is about applications, sessions, and state. By nature, the Web is a stateless medium—when you request a web page, the web server delivers it, and then forgets about you. If you want your web applications to recognize users when they make multiple page requests (for example, as they browse an e-commerce application adding items to a shopping basket), you need to know about the different techniques you can use to retain state across pages, for a session, or across an application.

- Chapters 7 and 8 focus on XML, a topic that has become very important as widespread Internet connectivity becomes the norm. In Chapter 7, we look at the concept of markup and how it is widely relevant to data-driven applications, and we create our own XML language by way of an XML Schema. In Chapter 8, we explore how to use that XML Schema to facilitate a data-transfer feature—exploiting XML's nature as the perfect vehicle for data transfer across the Internet.

- In Chapter 9, we turn briefly away from web sites to explore a different type of web application: the web service. Web services enable you to expose your application's functionality to other "client" applications. These applications make requests using standards and protocols over the Internet. This also means that you can use other people's web services in your code as if they were components on your own system, even though they are only accessible across the Internet. We'll examine both how to create web services and how to consume existing services.

- Chapter 10 is about ASP.NET authentication, authorization, and security. The role of security in an application is motivated by the need to restrict a user's (or application's) access to certain resources and ability to perform certain actions. For example, you may want to include administrative tools in your web application and to prevent access to these administrator pages for all but authorized users. This chapter looks at the ASP.NET tools for authenticating and authorizing users of your applications.

- Chapter 11 tackles two distinct but related subjects: debugging and exception handling. Debugging is much easier when you understand the different types of bugs that can occur, and easier still with the array of debugging tools and techniques made available by VS .NET and the .NET Framework. We'll study all that in the first half of the chapter. In the second half, we use the .NET exception mechanism to handle some potential input errors that could occur at runtime and prevent the application from crashing in a heap.

- In Chapter 12, we focus on two more different but related subjects: performance and caching. We set out to clarify what we mean by "good performance," and suggest a number of techniques you can use to analyze your application in realistic conditions. We'll apply some of those techniques to our Friends Reunion application, putting the application under stress to see what happens, and identifying and fixing a number of

different bottlenecks. We'll explain the issues related to caching and employ some caching techniques to save our application some processing effort, thereby optimizing the use of the server's resources.

- In Chapter 13, we describe how to prepare your application for deployment. VS .NET provides some easy-to-use tools that enable you to build your own deployment wizards. We demonstrate how to prepare the application for deployment—web site, database, and all—by wrapping it all up into an easy-to-use installation wizard.
- Appendix A contains a brief overview of the structure and functionality of the Friends Reunion web application, the design of the database, and the structure and use of the downloadable code.
- Appendix B contains more information about the setup and configuration of the Internet Information Server (IIS) web server and the Microsoft SQL Server Desktop Engine (MSDE).

What You Need to Use This Book

The following is the list of recommended system requirements for running the code in this book:

- A suitable operating system—server versions, such as Windows 2000 Server or Windows Server 2003 Web Edition, or professional versions, such as Windows 2000 or Windows XP Professional Edition
- Internet Information Server (IIS), which is shipped with the suitable operating systems (the version will depend on the operating system, but all of them are suitable for ASP.NET development)
- Visual Studio .NET (or Visual Basic .NET) Standard Edition or higher
- The Microsoft SQL Server Desktop Engine (MSDE) or Microsoft SQL Server

---

Note Windows XP Home Edition does *not* come with IIS and cannot run IIS. For ASP.NET web development on Windows XP Home Edition, you may consider the ASP.NET Web Matrix tool, available for free download from `http://www.asp.net`. This tool offers limited ASP.NET web server functionality, and you won't be able to run web projects in VS .NET with this version.

Read Appendix B for details on how to download and install MSDE if you don't have a full SQL Server. VS .NET provides a useful user interface for any SQL Server or MSDE database.

CHAPTER 1

Environment and Architecture

Windows desktop applications and web applications have many differences, but one difference is fundamental. This difference lies in the relative locations of the *application* itself and its *user interface*:

- When you run a Windows desktop application, the user interface appears on the screen of the machine on which the application is running. Messages between the application and its user interface are passed through the operating system of the machine. There's just one machine involved here and no network.
- When you run a web application, the user interface can appear in the browser of *any* machine. Messages between the application and its user interface must also pass across a network, because, typically, the web application and its user interface are on two *separate* machines.

This single difference in architecture manifests itself in many ways. If you're used to writing desktop applications and you're coming to web applications for the first time, it brings many new issues for you to consider. Let's begin with an overview of these considerations.

Web Application Considerations

Arguably, the most significant advantage of web applications is that the end users don't need to be on the same machine on which the application is running. In fact, they don't even need to be in the same country! But there are many other technical, practical, and design considerations, such as these:

Messaging: Since a running web application must communicate with its user interface across a network, there needs to be a way of passing messages between the two that is "network-proof."

Manipulating the user interface: How can a web application tell its browser-based user interface which buttons, text, labels, and so on to show, and how to arrange and style them?

Security: If a web application is available across a public network, you need to prevent unwanted users from accessing the application or from tapping in on authorized users.

Multiple users: A web application can be executed via a remote machine, so it can effectively be executed by two or more users at the same time (potentially millions of them if it's a successful one!).

Identification and state: How does a web application identify a user for the first time and recognize that user when she returns? This is especially important because of the stateless nature of the Web (see the "An Introduction to State Management" section later in this chapter).

If you're migrating from desktop application development to the Web, these are just a few issues that derive from the simple fact that an executing web application is (usually) physically separate from its user interface. We'll address all of the issues over the course of the book.

This chapter focuses on the web environment and on the architecture of web applications, to give you an idea of the implications of having an application and its user interface on different machines. In this chapter, we'll look at these aspects:

- How the web works, from the time the user requests a page to the time the user receives the response
- What a web application is and how it is composed
- The purpose of HTTP (the protocol underlying the Web) and its role in the request/response interaction between a browser and the web application server
- The role of the web server in hosting a web application
- The use of virtual directories in organizing web applications
- The difference between static content and dynamic content
- How client-side code and server-side code bring different effects to the world of dynamic content

We'll start by taking a look at how the web works and how requests for web pages are processed.

The Web Model

You can take advantage of the Web to set up an application that runs in one central location and can be accessed by users located anywhere in the world, through just a browser and an Internet connection.

The earliest web applications weren't really "applications" in the functional sense, but they took advantage of this basic concept to host documents in a single, central location and enable users to access those documents from distant places. With the global explosion of interest in the Internet, developers all over the world are now creating web applications that are much more functionally rich in their design.

Web applications no longer exist just as a central resource for shared documents. Now, they're still a central resource, but we use them interactively to buy groceries, to calculate our taxes, and to send and receive e-mail. Our children use the Web as an exciting, interactive

learning experience. We're now using web applications to perform all those interactive tasks that were previously only in the domain of the desktop application. We don't need to install the software anymore; we just need to point the browser across the Internet to the application.

Desktop Applications vs. Web Applications

If you've built a Windows desktop application in a language like Visual Basic .NET (VB .NET), then it's not too difficult to sit down and write a VB .NET *web* application that looks and feels quite similar to that desktop application, as illustrated in Figure 1-1. You can design the forms in the user interface (UI) to be similar, have the forms react to mouse clicks and button presses in a similar way, and make the back-end processing of the two applications quite similar.

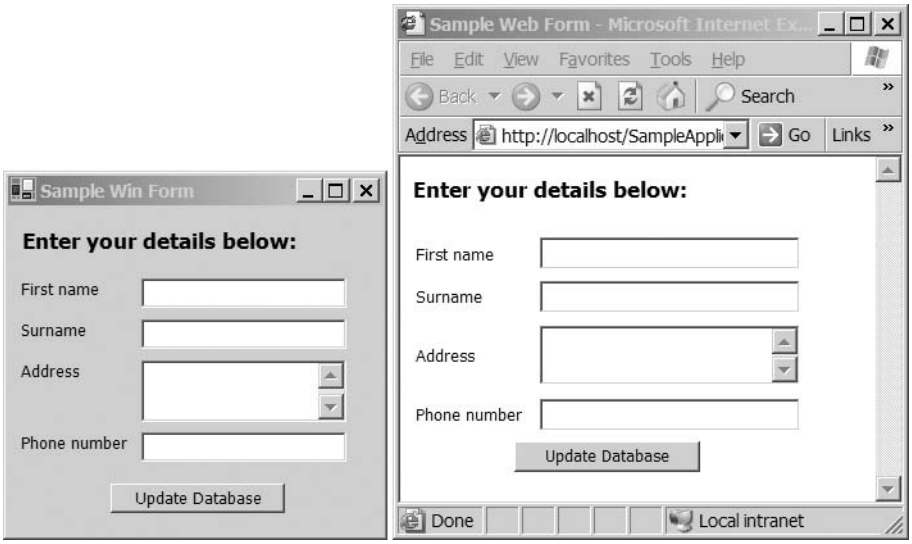

Figure 1-1. *A desktop application (left) and a web application (right) can look and feel quite similar.*

To use the desktop version of an application, you need to install it on a machine, and then sit at that machine while you use it. When you ask the operating system to run the application, it will create a new *process* for it, where the application will execute. Afterwards, your mouse clicks and keypresses will be detected by the operating system and passed to the process in which the desktop application is running. The application interprets these messages, does whatever processing is necessary, and tells the operating system what changes should be made to the UI as a result. All the communication between the application process and the UI is done via the operating system of that machine.

To use the web version of an application, you need to use a web browser. You type a URL into the web browser, which tells it where to find the machine on which the application is running. The browser arranges for a message to be sent across a network to this other machine. When the message is received by that machine, it's passed to the application, which interprets the message and decides what the UI (a web page) should look like. Then it sends a description of that web page back across the network to the browser.

Far more players are involved in the use of the web application than are involved with the desktop application, but the advantages of this remote-access concept are nothing short of phenomenal. To build an effective web application, you need to understand these interactions between the browser and web application in finer detail, because they shape the way you write web applications.

Web Servers and Web Clients

You know that by entering a URL into the Address box of a browser window, you can surf and navigate to a web site. But what actually happens, behind the scenes, when you press Enter or click the browser's Go button to submit that URL?

Before looking into the process, you need to understand the distinctions between the different machines and messages involved:

Web client/web server: The machine on which a browser is running is referred to as a *web client*. The machine on which the web application is running is called a *web server*.

Request/response: When a user types a URL into a browser, or clicks a link in a web page, the resulting message sent from the browser to the web server is called a *request*. The message containing the contents of a web page (which is sent by the web server to the browser in reaction to receiving the request) is called a *response*.

The *request* and *response* are the vital components of this communication process. From the point of view of a web application, they're critically important. The request tells the web application what to do, and the response contains the fruits of its labors.

With those concepts clearly defined, let's examine the process that takes place when a user employs a browser to request a web page. As illustrated in Figure 1-2, it's all about the exchange of messages: the request message and response message. In fact, the client is also able to send responses in addition to requests. The following sections describe each step in the process that you see in Figure 1-2.

Step 1: Initiating the Client Request

In the first stage, the user clicks a link, types a URL into a browser, or performs some similar action that initiates a request. The *request* is a message that must be sent to the web server.

In order to send any request (or response) message, the browser needs to do three things:

- *Describe* the message in a standard way so that it can be understood by the web server that receives it. For this, it uses the Hypertext Transfer Protocol (HTTP), the protocol used by the Web to describe both requests and responses. The described request message is called the *HTTP request*, and it has a very particular format that contains information about the request plus the information required to deliver it to the web server.
- *Package* the message so that it can be safely transported across the network. For this, it uses the Transmission Control Protocol (TCP).
- *Address* the message, to specify the place to which the message should be delivered. For this, it uses the Internet Protocol (IP).

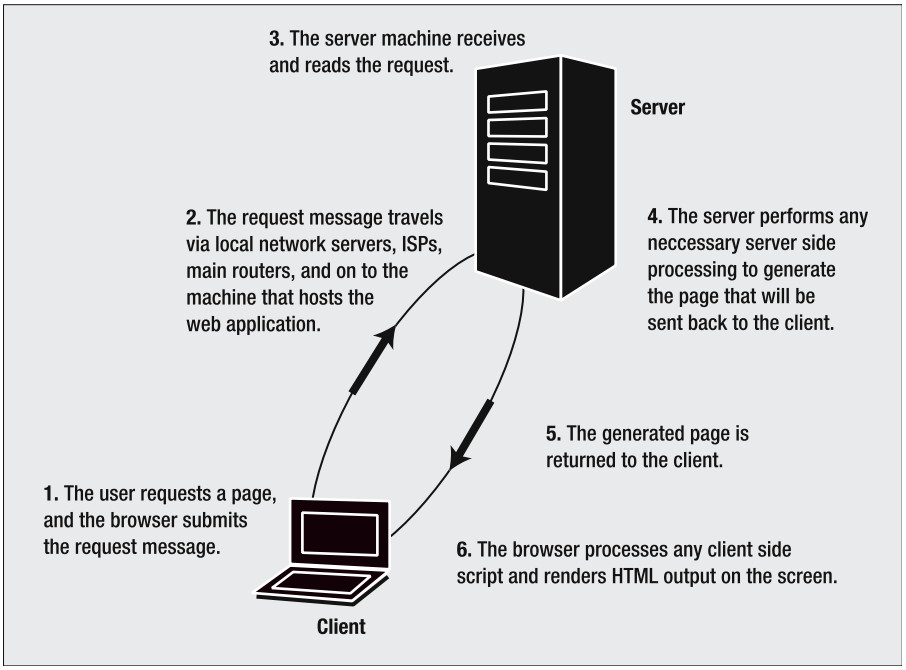

Figure 1-2. *What happens when a user requests a web page from a browser*

■ **Note** TCP and IP are often grouped together and referred to as TCP/IP. When you hear people talking about TCP/IP, they're discussing packaging and addressing Internet messages.

After the browser has described the message, packaged it, and addressed it, the request is ready to be dispatched across the network to its intended target: the web server.

Step 2: Routing the Request

Thanks to the HTTP, TCP, and IP protocols, the request message is formatted in such a way that it can be understood by each of the machines involved in *routing* the request—that is, passing the request from one machine to another as it finds its way from the web client to the web server.

The web server machine will be connected to the Internet (either directly or via a firewall) and will be uniquely identified on the Internet by its *IP address*. An IP address is a set of four numbers, each of which ranges between 0 and 255.

However, the original request probably didn't contain an IP address. More likely, it was made using a URL that began with a named web site address (this is usually something more memorable, like `http://www.apress.com/`). The link between an IP address and its named equivalent is mapped using the Domain Name Service (DNS). For example, DNS currently

maps the URL `http://www.microsoft.com/` to the IP address `http://207.46.245.222/`, so requests for web pages at the URL `http://www.microsoft.com/` are directed to the web server whose IP address is `http://207.46.245.222/`.

■ **Note** The set of four numbers in an IP address enables us to address almost 4.3 billion machines uniquely. IP version 6 (IPv6) is a new version of the protocol that allows for many more unique addresses by employing eight sets of four hexadecimal digits (so that each set of four is between 0000 and FFFF). Despite the fact that 4.3 billion sounds like a lot, the invention of portable devices with "always on" Internet connections, such as mobile phones connecting over General Packet Radio Service (GPRS), means that we will eventually run out of today's IP addresses. This is why IPv6 was created.

One address you'll be seeing a lot of in this book is `http://localhost/`. This is a special address that resolves to the IP address `http://127.0.0.1/`. Any computer recognizes this address as referring to itself. It's not actually a real IP address, but a reserved address used as a shortcut for the local machine, known as a *loopback address*.

Step 3: Receiving and Reading the HTTP Request

The standardized format for web requests is defined by HTTP, so when the HTTP request arrives at its destination, the web server knows exactly how to read it.

In HTTP, a web client can make two principal types of requests of a server:

`GET` request: The client can ask the server to send it a resource such as a web page, a picture, or an MP3 file. This is called a `GET` request, because it gets information from the server. This is a commonly used method, for example, when developing a search facility on a web site, where the same request will be made on more than one occasion. It's also how you request simple web pages and images.

`POST` request: The client can ask the server to perform some processing in order to generate a response. This is called a `POST` request, because the client posts the information that the server must process, and then awaits a response. This method is more likely to be used, for example, when submitting personal data to an online shopping site or in any other situation where you are sending information to the server.

Step 4: Performing Server-Side Processing

The web server is the place where the application is running. It is responsible for ensuring that any necessary server-side processing takes place in order to complete its task and generate a response.

If the HTML request contains a request for a simple HTML page, the web server will locate the HTML page, wrap it into an HTTP response, and dispatch it back to the client. In contrast, if the request is for an `.aspx` page, for example, the web server will pass the request to the ASP.NET engine, which takes care of processing the page and generating the output (usually HTML), before the web server wraps that newly generated output into an HTTP response, ready to be sent back to the client.

The HTTP response consists of two parts: the header and the body. The *header* contains information that tells the browser whether the request was successful or there was an error processing it. For example, if the web server received the request but was unable to locate the requested page, the header will contain an HTTP 404 (File Not Found) error code. The response *body* is where a successfully requested resource (such as a block of HTML) is placed.

Step 5: Routing the Response

The HTML page, generated at the web server, has been described in terms of an HTTP response message, and is packaged and addressed using TCP/IP. The return address is an IP address, which was included in the HTTP request message sent in step 1.

The HTTP response message is routed back across the network to its target destination: the web client that made the original request.

Step 6: Performing Client-Side Processing and Rendering

When the HTTP response reaches the web client, the browser reads the response and processes any client-side code. This processed code is now displayed in the browser window.

Now that you know what goes on when a user requests a page, it's time to start using some code.

System Configuration for Web Development

The best way to understand the web application process is to create some web pages and look at them from the perspective of both the end user and web server. You'll begin to do this shortly (and you'll create plenty of web pages during the course of the book), but before you can start, you need to install and configure a web server, which you'll use to host your web applications.

If you are running Windows 2000, Windows XP, or even Windows Server 2003, you'll be able to use Microsoft's own web server—Internet Information Services (IIS)—to host your applications. Other web servers are available for Windows (such as Cassini, a web server written in .NET) and for other development platforms (notably Apache, which can be installed on Windows machines, Linux machines, and now even comes preinstalled on Mac OS X machines). For this book, we'll use IIS to host the sample web applications, because it's the platform provided by Microsoft in order to run ASP.NET applications (although not the only one).

Different versions of IIS are supplied as part of different versions of the Windows operating system:

- Windows 2000 Professional or higher has IIS 5.0
- Windows XP Professional has IIS 5.1
- Windows Server 2003 has IIS 6.0

IIS 5.0 and IIS 5.1 are very similar (in fact, IIS 5.1 is just a minor update of IIS 5.0). IIS 6.0 is a more substantial revamping of the IIS architecture, providing improvements in security and speed, as well as a deeper level of integration with ASP.NET applications.

Having said that, the front-end interface for all of these versions is very similar; therefore, the configuration process described in this chapter works equally well for all of these IIS

versions. For the remainder of the book, we'll limit the discussions mainly to the features available to IIS 5.*x* developers, because it's the main platform for workstations. We've included an overview and setup instructions for IIS 6.0 in Appendix B, as there are some differences that should be taken into account when setting up web development or deploying the application on a server with Windows Server 2003.

Installing and Configuring IIS for .NET Web Applications

The IIS installation process itself is a fairly painless operation. If you haven't installed any of the necessary software yet, then it's recommended that you install IIS *before* either the .NET Framework or Visual Studio .NET (VS. NET), because the installation of the latter configures the former so that it is able to deal with the ASP.NET files that are central to web applications in .NET.

If You Haven't Installed the .NET Framework or VS .NET Yet

If you have not yet installed the .NET Framework or VS. NET, open the Control Panel and head to Add/Remove Programs. In the dialog box that appears, select Add/Remove Windows Components. In the list of components that appears, select Internet Information Services (IIS), as shown in Figure 1-3. Accept all the default settings, and the installation process will commence. Now, you can go ahead and install the .NET Framework and VS. NET, too.

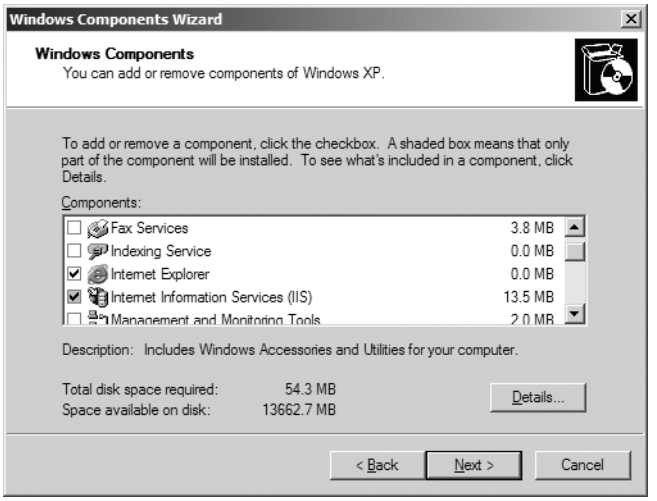

Figure 1-3. *Choosing to install IIS*

If You've Already Installed the .NET Framework or VS .NET

If you have already installed the .NET Framework and/or VS .NET, you could proceed by uninstalling VS .NET, and then uninstalling the .NET Framework (via the Add/Remove Programs group in your Control Panel). Then you could install IIS, and finally, reinstall VS .NET. But this is an aggressive (and time-consuming) process. Instead, we recommend that you try to use one of the following tools to "repair" the .NET Framework, and use the uninstall/reinstall method only as a last resort.

First, try to run the ASP.NET IIS registration utility. Select Start ➤ Run. In the dialog box that appears, type the following command:

```
%systemroot%\Microsoft.NET\Framework\[Version]\aspnet_regiis.exe
```

`[Version]` is the .NET version you have: `v1.0.3705` for .NET 1.0 or `v1.1.4322` for .NET 1.1.

If that doesn't work, you should be able to do the job using the VS. NET DVD or CDs.

If you own the VS .NET DVD, insert the DVD, and then select Start ➤ Run. In the dialog box that appears, type the following command (on one line):

```
<Drive>:\wcu\dotNetFramework\dotnetfx.exe
                    /t:c:\temp
                    /c:"msiexec.exe
                    /fvecms c:\temp\netfx.msi"
```

If you have the VS .NET CDs, insert the VS .NET Windows Component Update CD, and then select Start ➤ Run. In the dialog box, type the following command (on one line):

```
<Drive>:\dotNetFramework\dotnetfx.exe
                          /t:c:\temp
                          /c:"msiexec.exe
                          /fvecms c:\temp\netfx.msi"
```

Administering IIS

Once the IIS installation has completed, you can check that it was successful by viewing the IIS console. The IIS console is the key administration tool that you will use to configure your IIS web server software.

To run the console, select Start ➤ Run, type **inetmgr** in the dialog box, and click OK. Alternatively, you can use the Internet Services Manager shortcut in the Administrative Tools group in your Control Panel (if you are running Windows XP, this link is named Internet Information Services). The IIS console appears, and it should look something like the window shown in Figure 1-4.

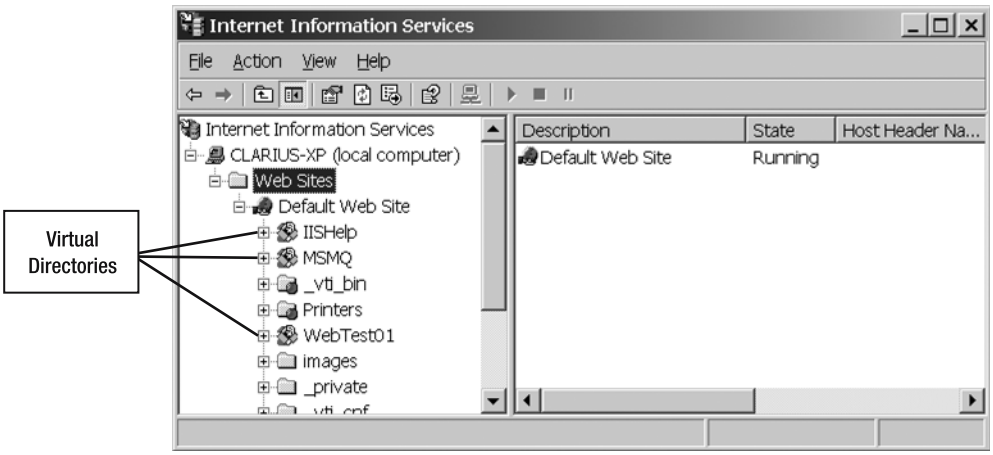

Figure 1-4. *IIS console is an administration tool for configuring IIS web server software.*

Creating Virtual Directories

In the folder list, you will notice that several of the folders are marked with a package-like (gear-like in IIS 6.0) icon (shown in Figure 1-4). This icon indicates that the folder is a *virtual directory*, which is also configured to be a root location for a web application.

IIS maps each virtual directory to a physical directory on the web server machine. This mapping works as follows:

- The virtual directory structure is the structure that the end users see when they browse the pages of the web site though a browser.
- The physical directory structure is the *actual* folder structure that is used on the web server's hard disk for organizing and storing the pages and resources of the web application.

Thus, web developers can use this mapping of virtual directories to physical ones to place the web application's source files in the most convenient place on their server and to hide their server's physical organization from end users (each virtual directory is just the alias of a physical directory). Let's see how this works and examine the process in more detail in an example.

Try It Out: Create a Virtual Directory In this example, you'll use the IIS console to create a virtual directory that points to a physical folder on your system. You'll write a simple HTML page and place it inside the physical directory, so that it will be possible to use a browser to navigate to that page via HTTP, using the virtual directory structure you created.

1. Use Windows Explorer to create a directory called `Apress` anywhere on your drive, and then create a subdirectory called `BegWeb` within it, so that you have a physical directory structure like `C:\Apress\BegWeb`.
2. Fire up a text or code editor such as Notepad. (You'll be using VS .NET in every other example in the book, but for this example, it's really not necessary.) Enter the following simple HTML code into a new file:

```
<html>
  <head>
    <title>Beginning Web Programming - Simple Test HTML Page</title>
  </head>
  <body>
    <h2>A Simple HTML Page</h2>
    <p>Hello world!</p>
  </body>
</html>
```

3. Save this file as `HelloWorld.htm` in the `\BegWeb` directory that you just created.
4. Now, you'll create a virtual directory. Open the IIS console (select Start ➤ Run and type **inetmgr**). Right-click the Default Web Site node, and select New ➤ Virtual Directory from the context menu, as shown in Figure 1-5.

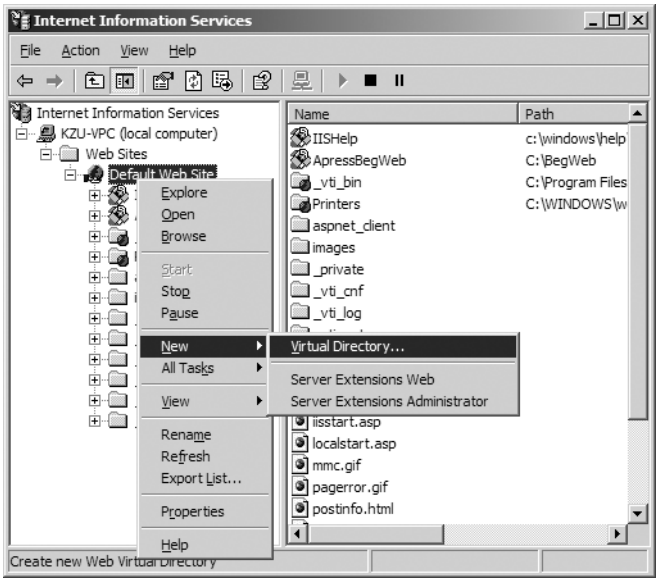

Figure 1-5. *Choosing to create a new virtual directory*

5. At this point, an introductory dialog box will appear. Just click Next to continue.

6. In the first page of the wizard, you'll be prompted for the name that you want to give to your virtual directory. This is the name that end users will see in the Address box of a browser when they request the page. Name your directory `ApressBegWeb`, as shown in Figure 1-6. Then click Next.

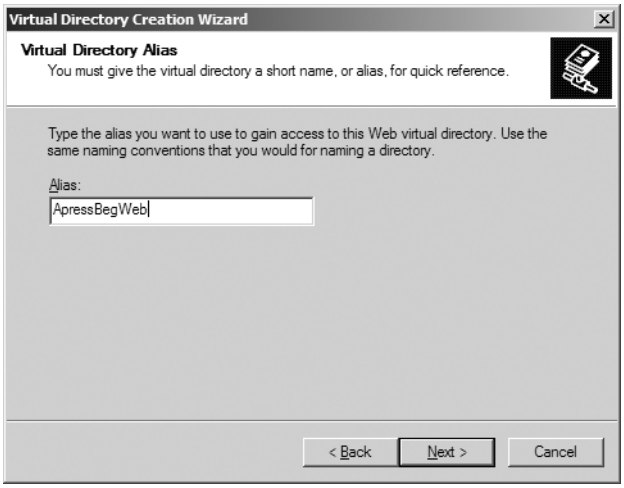

Figure 1-6. *Naming your virtual directory*

7. In the next wizard page, browse to the `BegWeb` physical directory (or just enter the directory path into the text box), as shown in Figure 1-7, and then click Next.

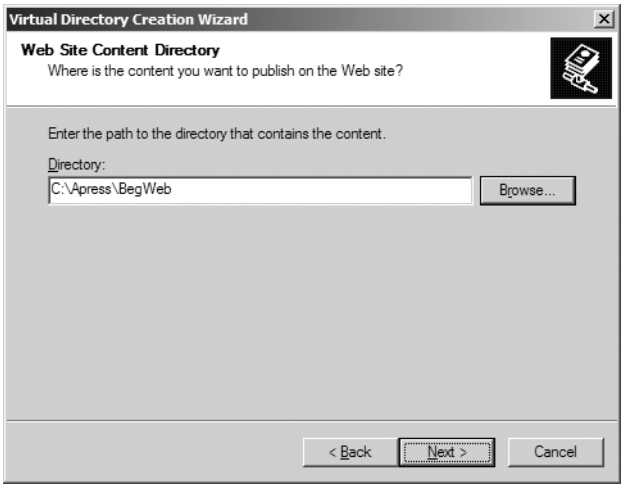

Figure 1-7. *Specifying the physical directory that contains the web page contents*

8. The next page, Access Permissions, presents a number of options relating to the permissions enabled on resources contained within this virtual directory, as shown in Figure 1-8. By default, users will be able to read files and run script-based programs. For now, leave the default settings and click Next to continue.

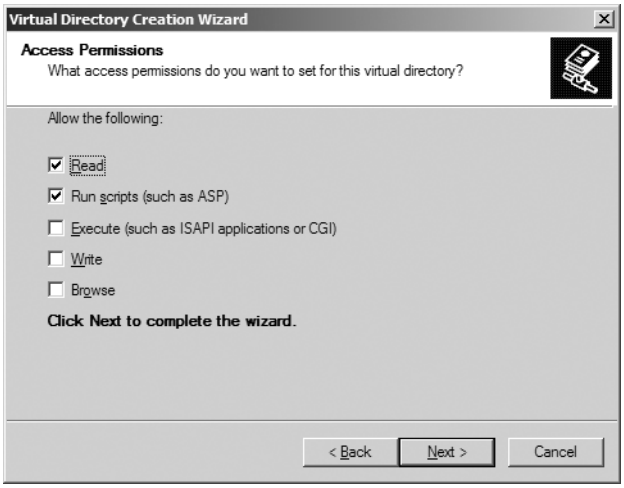

Figure 1-8. *Setting access permissions*