sitepoint®

BUILD YOUR OWN ASP.NET 4 WEBSITE

USING C# & VB

BY **CRISTIAN DARIE**
WYATT BARNETT
& **TIM POSEY**

4TH EDITION

THE ULTIMATE ASP.NET BEGINNER'S GUIDE

Summary of Contents

BUILD YOUR OWN ASP.NET 4 WEBSITE USING C# & VB

BY **CRISTIAN DARIE**
WYATT BARNETT
TIM POSEY
4TH EDITION

Build Your Own ASP.NET 4 Website Using C# & VB

by Cristian Darie, Wyatt Barnett, and Tim Posey

Copyright © 2011 SitePoint Pty. Ltd.

Expert Reviewer: Pranav Rastogi
Product Editor: Simon Mackie
Technical Editor: Ricky Onsman
Printing History:
First Edition: April 2004
Second Edition: October 2006
Third Edition: September 2008
Editor: Sarah Broomhall
Index Editor: Michelle Combs
Cover Design: Alex Walker
Latest Update: Fourth Edition: September 2011

Notice of Rights

All rights reserved. No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

Notice of Liability

The author and publisher have made every effort to ensure the accuracy of the information herein. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors and SitePoint Pty. Ltd., nor its dealers or distributors, will be held liable for any damages to be caused either directly or indirectly by the instructions contained in this book, or by the software or hardware products described herein.

Trademark Notice

Rather than indicating every occurrence of a trademarked name as such, this book uses the names only in an editorial fashion and to the benefit of the trademark owner with no intention of infringement of the trademark.

Published by SitePoint Pty. Ltd.

48 Cambridge Street Collingwood
VIC Australia 3066.
Web: www.sitepoint.com
Email: business@sitepoint.com

ISBN 978-0-9870908-6-7 (print)

ISBN 978-0-9871530-3-6 (ebook)
Printed and bound in the United States of America

About the Authors

Cristian Darie is a software engineer with experience in a wide range of modern technologies, and the author of numerous technical books, including the popular *Beginning E-Commerce* series. He initially tasted programming success with a prize in his first programming contest at the age of 12. From there, Cristian moved on to many other similar achievements, and is now studying distributed application architectures for his PhD.

Wyatt Barnett leads the in-house development team for a major industry trade association in Washington DC. When not slinging obscene amounts of C# and SQL at a few exceedingly large monitors, he is most often spotted staring at HDTV and other forms of entertainment in local watering holes.

Tim Posey is a long-time developer and a passionate educator. Armed with a B.S. in Computer Science and an M.B.A. in Finance, he has traversed many industries, consulting for multiple corporations in banking, insurance, energy, and various e-commerce industries. As a serial entrepreneur, he mentors local startups and non-profit organizations. He serves as a senior software engineer at a Fortune 1000 company and an Adjunct Professor of Finance for the American Public University System. His favorite pastime is watching Alabama football. He may be contacted at tim@timposey.net.

About the Technical Editor

Ricky Onsman is an Australian freelance web designer and jack of all trades. With a background in information and content services, he built his first website in 1994 for a disability information service and has been messing about on the web ever since. He is the president of the Web Industry Professionals Association.

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit http://www.sitepoint.com/ to access our books, newsletters, articles, and community forums.

To my family and friends.

—Cristian Darie

To my Father, whose guidance got me this far.

—Wyatt Barnett

For LJ and Erin.

—Tim Posey

Table of Contents

Foreword

Before you go much further in reading this book, give yourself a small pat on the back for investing the money, time and effort in learning ASP.NET. Perhaps it is a new technology to you, or perhaps you are familiar with ASP or other programming in .NET. Either way, it's a great skill to add to your toolbox and increase your value as a developer.

ASP.NET is useful in more ways than one. If you aren't already a .NET developer, it's the gateway to learning the framework, and the languages that you can use to program against it. The most common languages, and the ones covered in this book, are C# and VB.NET. Skills in these languages and framework go way beyond web development. You can use them for mobile development with Silverlight, which uses the .NET framework for Windows Phone 7 Desktop development; or .NET on Windows Power Desktop development with the Windows Presentation Foundation (WPF), part of the .NET Framework Workflow development for business processes using the Workflow Foundation (WF)—which is also part of the .NET Framework Connected systems development using the Windows Communication Foundation (WCF).

Beyond these, the skills continue to grow in relevance as the industry matures and develops. Time invested in .NET development will reap benefits with cloud-scalable applications using Windows Azure, as well as the new Windows 8 client applications. But you have to start somewhere, and starting with the web is a wise choice. ASP.NET allows you to build dynamic websites, web applications and web services. As a developer, you know and understand that there as many different types of web application as there are web applications themselves, and you need a powerful and flexible framework that will allow you to build them, without having to reinvent the wheel each time.

ASP.NET is this framework, and with its Web Forms and Controls technologies, you can use rapid development methodologies to get your application up and running quickly. Being fully standards-compliant, you can also make it beautiful using CSS. Beyond this, particularly for professional, commercial applications, you'll need tools that allow database connectivity to be smart, secure, and efficient, and ASP.NET with its ADO.NET technology provides this for you.

And of course it wouldn't be Web 2.0 if you didn't have the ability to use Ajax. ASP.NET gives you simple but effective ways to use AJAX with server-side controls that do a lot of the hard work of handling asynchronous page updates for you. Indeed, server-side coding is something that you'll do a lot of with ASP.NET. It's amazing how simple it can make writing distributed applications, where the server is smart enough to manage sessions, connectivity, presentation and more on your behalf.

This book provides you with everything you need to know to skill up in ASP.NET development with Web Forms technology. It's a fantastic learning tool, written in an approachable and informative way. I strongly recommend you pick up your copy of this book, download the free Visual Web Developer Express tools, and start coding in ASP.NET. You'll be amazed at what you can build, quickly and easily.

Laurence Moroney, technologist and author

August 2011

Preface

Web development is very exciting. There's nothing like the feeling you have after you place your first dynamic web site online, and see your little toy in action while other people are actually using it!

Web development with ASP.NET is particularly exciting. If you've never created a dynamic web site before, I'm sure you'll fall in love with this area of web development. If you've worked with other server-side technologies, I expect you'll be a little shocked by the differences.

ASP.NET really is a unique technology, and it provides new and extremely efficient ways to create web applications using the programming language with which you feel most comfortable. Though it can take some time to learn, ASP.NET is simple to use. Whether you want to create simple web forms, feature-rich shopping carts, or even complex enterprise applications, ASP.NET can help you do it. All the tools you'll need to get up and running are immediately available and easy to install, and require very little initial configuration.

This book will be your gentle introduction to the wonderful world of ASP.NET, teaching you the foundations step by step. First, you'll learn the theory; then, you'll put it into practice as we work through practical exercises together. Finally, we'll stretch your abilities by introducing the MVC Framework and other advanced topics. To demonstrate some of the more complex functionality, and to put the theory into a cohesive, realistic context, we'll develop a project through the course of this book. The project—an intranet site for a company named Dorknozzle—will allow us to see the many components of .NET in action, and to understand through practice exactly how .NET works in the real world.

We hope you'll find reading this book an enjoyable experience that will significantly help you with your future web development projects!

Who Should Read This Book

This book is aimed at beginner, intermediate, and advanced web designers looking to make the leap into server-side programming with ASP.NET. We expect that you'll already feel comfortable with HTML, CSS, and a little knowlegable about database

design although we will cover quite a few databatse topics along the way. Developers in open-source web development languages such as PHP, Java, or Ruby will make an excellent transition to learning ASP.NET.

By the end of this book, you should be able to successfully download and install Visual Web Developer 2010 Express Edition, and use it to create basic ASP.NET pages. You'll also learn how to install and run Microsoft SQL Server 2008 R2 Express Edition, create database tables, and work with advanced, dynamic ASP.NET pages that query, insert, update, and delete information within a database.

All examples provided in the book are written in both Visual Basic and C#, the two most popular languages for creating ASP.NET websites. The examples start at beginners' level and proceed to more advanced levels. As such, no prior knowledge of either language is required in order to read, understand, learn from, and apply the knowledge provided in this book. Experience with other programming or scripting languages (such as JavaScript) will certainly grease the wheels, though, and should enable you to grasp fundamental programming concepts more quickly.

What's in This Book

This book comprises the following chapters. Read them from beginning to end to gain a complete understanding of the subject, or skip around if you feel you need a refresher on a particular topic.

Chapter 1: Introducing ASP.NET

Before you can start building your database-driven web presence, you must ensure that you have the right tools for the job. In this first chapter, you'll install Visual Web Developer 2010 Express Edition and Microsoft SQL Server 2008 R2 Express Edition. Finally, you'll create a simple ASP.NET page to make sure that everything's running and properly configured.

Chapter 2: ASP.NET Basics

In this chapter, you'll create your first useful ASP.NET page. We'll explore all the components that make up a typical ASP.NET page, including directives, controls, and code. Then, we'll walk through the process of deployment, focusing specifically on allowing the user to view the processing of a simple ASP.NET page through a web browser.

Chapter 3: VB and C# Programming Basics

In this chapter, we'll look at two of the programming languages that are used to create ASP.NET pages: VB and C#. You'll learn about the syntax of the two languages as we explore the concepts of variables, data types, conditionals, loops, arrays, functions, and more. Finally, we'll see how these languages accommodate object oriented programming principles by allowing you to work with classes, methods, properties, inheritance, and so on.

Chapter 4: Constructing ASP.NET Web Pages

ASP.NET web pages are known as web forms, but, as we'll see, the process of building ASP.NET web forms is a lot like creating a castle with Lego bricks! ASP.NET is bundled with hundreds of controls—including HTML controls, web controls, and so on—that are designed for easy deployment within your applications. This chapter will introduce you to these building blocks and show how to lock them together. You'll also learn about master pages, which are a very exciting feature of ASP.NET.

Chapter 5: Building Web Applications

A web application is basically a group of web forms, controls, and other elements that work together to achieve complex functionality. So it's no surprise that when we build web applications, we must consider more aspects than when we build individual web forms. This chapter touches on those aspects. You'll configure your web application; learn how to use the application state, user sessions, and cookies; explore the process for debugging errors in your project; and more.

Chapter 6: Using the Validation Controls

This chapter introduces validation controls. With validation controls, Microsoft basically eliminated the headache of fumbling through and configuring tired, reused client-side validation scripts. First, you'll learn how to implement user input validation on both the client—and server sides—of your application using Microsoft's ready-made validation controls. Then, you'll learn how to perform more advanced validation using regular expressions and custom validators.

Chapter 7: Database Design and Development

Undoubtedly, one of the most important chapters in the book, Chapter 7 will prepare you to work with databases in ASP.NET. We'll cover the essentials you'll need to know in order to create a database using SQL Server 2008 R2

Express Edition. As well, you'll begin to build the database for the Dorknozzle intranet project.

Chapter 8: Speaking SQL

This chapter will teach you to speak the language of the database: Structured Query Language, or SQL. After a gentle introduction to the basic concepts of SQL, which will teach you how to write `SELECT`, `INSERT`, `UPDATE`, and `DELETE` queries, we'll move on to more advanced topics such as expressions, conditions, and joins. Finally, we'll take a look at how you can reuse queries quickly and easily by writing stored procedures.

Chapter 9: ADO.NET

The next logical step in building database-driven web applications is to roll up our sleeves and dirty our hands with a little ADO.NET—the technology that facilitates communication between your web application and the database server. This chapter explores the essentials of the technology, and will have you reading database data directly from your web applications in just a few short steps. You'll then help begin the transition from working with static applications to those that are database driven.

Chapter 10: Displaying Content Using `DataLists`

Taking ADO.NET further, this chapter shows you how to utilize the `DataList` control provided within the .NET Framework. `DataLists` play a crucial role in simplifying the presentation of information with ASP.NET. In learning how to present database data within your applications in a cleaner and more legible format, you'll gain an understanding of the concepts of data binding at a high level.

Chapter 11: Managing Content Using `GridView` and `DetailsView`

This chapter explores two of the most powerful data presentation controls of ASP.NET: `GridView` and `DetailsView`. `GridView` is a very dynamic control that automates almost all tasks that involve displaying grids of data. `DetailsView` completes the picture by offering the functionality needed to display the details of a single grid item.

Chapter 12: Advanced Data Access

This chapter explores a few of the more advanced details involved in data access, retrieval, and manipulation. We'll start by looking at direct data access using

ADO.NET's data source controls. We'll then compare this approach with that of using data sets to access data in a disconnected fashion. In this section, you'll also learn to implement features such as paging, filtering, and sorting, using custom code.

Chapter 13: Security and User Authentication

This chapter will show you how to secure your web applications with ASP.NET. We'll discuss the various security models available, including IIS, Forms, Windows, and Windows Live ID, and explore the roles that the **Web.config** and XML files can play. This chapter will also introduce you to the ASP.NET membership model and login controls.

Chapter 14: Working with Files and Email

In this chapter, we'll look at the task of accessing your server's file system, including drives, files, and the network. Next, I'll will show you how to work with file streams to create text files, write to text files, and read from text files stored on your web server. Finally, you'll gain first-hand experience in sending emails using ASP.NET.

Chapter 15: Introduction to LINQ

Here we learn about LINQ, a language construct that allows us to query relational data from different sources and interact with it just like any other object or class. With LINQ we get access to compile-time syntax checking, the use of IntelliSense, and the ability to access other data sources such as XML or just about any custom data sources.

Chapter 16: Introduction to MVC

In this chapter we familiarise ourselves with the Model-View-Controller architecture to solve problems in software development and maintenance, separating our business logic, user interface and control flow.

Chapter 17: ASP.NET AJAX

In our final chapter, you'll learn all about the Ajax features that are built into ASP.NET 4. We'll spice up the Dorknozzle project with a few Ajax features that'll show how simple ASP.NET AJAX is to use. We'll also explore the ASP.NET AJAX Control Toolkit, and see how it can enhance existing features.

Appendix A: Web Control Reference

Included in this book is a handy web control reference, which lists the most common properties and methods of the most frequently used controls in ASP.NET.

Appendix B: Deploying ASP.NET Websites

Here you'll be shown, step by step, how to use Visual Web Developer and how to move your website from your development environment to a web hosting service and make it live on the Internet. It also covers tips for choosing a reliable web host, ASP.NET deployment gotchas, and hints for using the SQL Server Hosting Toolkit to migrate your database.

Where to Find Help

SitePoint had a thriving community of web designers and developers ready and waiting to help you out if you run into trouble. We also manintain a list of known errat for the book, which you can consult for the latest updates.

The SitePoint Forums

The SitePoint Forums are [1]discussion forums where you can ask questions about anything related to web development. You may, of course, answer questions too. That's how a forum site works—some people ask, some people answer, and most people do a bit of both. Sharing your knowledge benefits others and strenghtens the community. A lot of interesting and experienced web designers and developers hang out there. It's a good way to learn new stuff, have questions answered in a hurry, and generally have a blast.

The Book's Website

Located at http://www.sitepoint.com/books/aspnet4/, the website that supports this book will give you access to the following facilities:

The Code Archive

As you progress through this book, you'll note a number of references to the code archive. This is a downloadable ZIP archive that contains every line of example source code printed in this book. If you want to cheat (or save yourself from carpal

[1] http://www.sitepoint.com/forums/

tunnel syndrome), go ahead and download the archive. [2]The archive contains one folder for each chapter of this book. Each folder may contain a **LearningASP** folder for the stand-alone examples in that chapter and a **Dorknozzle** folder for files associated with the Dorknozzle intranet application, the project that we'll work on throughout the book. Each folder will contain **CS** and **VB** subfolders, which contain the C# and VB versions of all the code examples for that chapter. Incremental versions of each file are represented by a number in the file's name.

Updates and Errata

No book is perfect, and we expect that watchful readers will be able to spot at least one or two mistakes before the end of this one. The Errata page [3]on the book's website will always have the latest information about known typographical and code errors.

The SitePoint Newsletters

In addition to books like this one, SitePoint publishes free email newsletters including *The SitePoint Tribune* and *The SitePoint Tech Times*. In them, you'll read about the latest news, product releases, trends, tips, and techniques for all aspects of web development. If nothing else, you'll gain useful ASP.NET articles and tips, but if you're interested in learning other technologies, you'll find them especially valuable. You can subscribe at http://www.sitepoint.com/newsletter/.

The SitePoint Podcast

Join the SitePoint Podcast team for news, interviews, opinion, and fresh thinking for web developers and designers. We discuss the latest web industry topics, present guest speakers, and interview some of the best minds in the industry. You can catch up on the latest and previous podcasts at http://www.sitepoint.com/podcast/, or subscribe via iTunes.

Your Feedback

If you 're unable to find an answer through the forums, or if you wish to contact us for any other reason, the best place to write is `books@sitepoint.com`. We have a

[2] http://www.sitepoint.com/books/aspnet4/code.php
[3] http://www.sitepoint.com/books/aspnet4/errata.php

well-staffed email support system set up to track your inquiries, and if our support team members are unable to answer your question, they'll send it straight to us. Suggestions for improvements, as well as notices of any mistakes you may find, are especially welcome.

Acknowledgments

I'd like to thank the many folks at SitePoint, including Tom, Ricky, Sarah, and Simon, for giving me the opportunity for this book and helping to produce a magnificent product. Special thanks to Pranav Rastogi from Microsoft for giving me detailed technical insight into the many behind-the-scenes details and undocumented features to help make this book a success. Finally, I would like to extend special thanks to my wife for enduring many long nights of having to put our child to bed while I worked on this project.

—Tim Posey

Conventions Used in This Book

You'll notice that we've used certain typographic and layout styles throughout this book to signify different types of information. Look out for the following items.

Code Samples

Code in this book will be displayed using a fixed-width font, like so:

```
<h1>A Perfect Summer's Day</h1>
<p>It was a lovely day for a walk in the park. The birds
were singing and the kids were all back at school.</p>
```

If the code is to be found in the book's code archive, the name of the file will appear at the top of the program listing, like this:

example.css

```
.footer {
  background-color: #CCC;
  border-top: 1px solid #333;
}
```

If only part of the file is displayed, this is indicated by the word *excerpt*:

example.css *(excerpt)*

```
  border-top: 1px solid #333;
```

If additional code is to be inserted into an existing example, the new code will be displayed in bold:

```
function animate() {
  new_variable = "Hello";
}
```

Where existing code is required for context, rather than repeat all the code, a vertical ellipsis will be displayed:

```
function animate() {
  ⋮
  return new_variable;
}
```

Some lines of code are intended to be entered on one line, but we've had to wrap them because of page constraints. A ➥ indicates a line break that exists for formatting purposes only, and should be ignored:

```
URL.open("http://www.sitepoint.com/blogs/2007/05/28/user-style-she
➥ets-come-of-age/");
```

Tips, Notes, and Warnings

Hey, You!

Tips will give you helpful little pointers.

Ahem, Excuse Me ...

Notes are useful asides that are related—but not critical—to the topic at hand. Think of them as extra tidbits of information.

Make Sure You Always ...

... pay attention to these important points.

Watch Out!

Warnings will highlight any gotchas that are likely to trip you up along the way.

Chapter 1

Introducing ASP.NET and the .NET Platform

By now, ASP.NET is one of the most popular web development technologies on the planet. The first version was released in 2002, and since then, Microsoft has continued the tradition of releasing a powerful web development framework that allows web developers to do more with less. ASP.NET has experienced rapid growth among the established corporate world, as well as becoming the choice for many freelance developers. ASP.NET has many advantages, including a well-established IDE (Integrated Development Environment) called Microsoft Visual Studio, and advanced security and performance frameworks that handle many of the mundane tasks automatically on the server side, freeing the developer to create more full-fledged web applications and websites.

ASP.NET 4 is the latest iteration in the .NET framework, introducing many new features that build upon its predecessor to improve performance, security, and interoperability with the latest browsers. Best of all, it comes available with new development tools, including Visual Web Developer 2010 Express Edition and SQL

Server 2008 R2 Express Edition, both of which are free! These tools enable the rapid application development (RAD) of web applications.

The goal of this book is to enable you to use all these technologies together in order to produce fantastic results. We'll take you step by step through each task, showing you how to get the most out of each technology and tool. Let's begin!

What is ASP.NET?

ASP.NET is a sophisticated and powerful web development framework. If you've never used ASP.NET before, it's likely to take you some time and patience to grow accustomed to it. Development with ASP.NET requires not only an understanding of HTML and web design, but a firm grasp of the concepts of object oriented programming and development. Fortunately, we believe you'll find the benefits amply reward the learning effort!

In the next few sections, we'll introduce you to the basics of ASP.NET. We'll walk through the process of installing it on your web server, and look at a simple example that demonstrates how ASP.NET pages are constructed. But first, let's define what ASP.NET actually *is*.

ASP.NET is a server-side technology for developing web applications based on the Microsoft .NET Framework. Okay, let's break that jargon-filled sentence down.

ASP.NET is a server-side technology. That is, it runs on the web server. Most web designers cut their teeth learning client-side technologies such as HTML, JavaScript, and Cascading Style Sheets (CSS). When a web browser requests a web page created with only client-side technologies, the web server simply grabs the files that the browser (or client) requests and sends them down the line. The client is entirely responsible for reading the markup in those files and interpreting that markup to display the page on the screen.

Server-side technologies such as ASP.NET, however, are a different story. Instead of being interpreted by the client, server-side code (for example, the code in an ASP.NET page) is interpreted by the web server. In the case of ASP.NET, the code in the page is read by the server and used to generate the HTML, JavaScript, and CSS, which is then sent to the browser. Since the processing of the ASP.NET code occurs on the server, it's called a server-side technology. As Figure 1.1 shows, the

client only sees the HTML, JavaScript, and CSS. The server is entirely responsible for processing the server-side code.

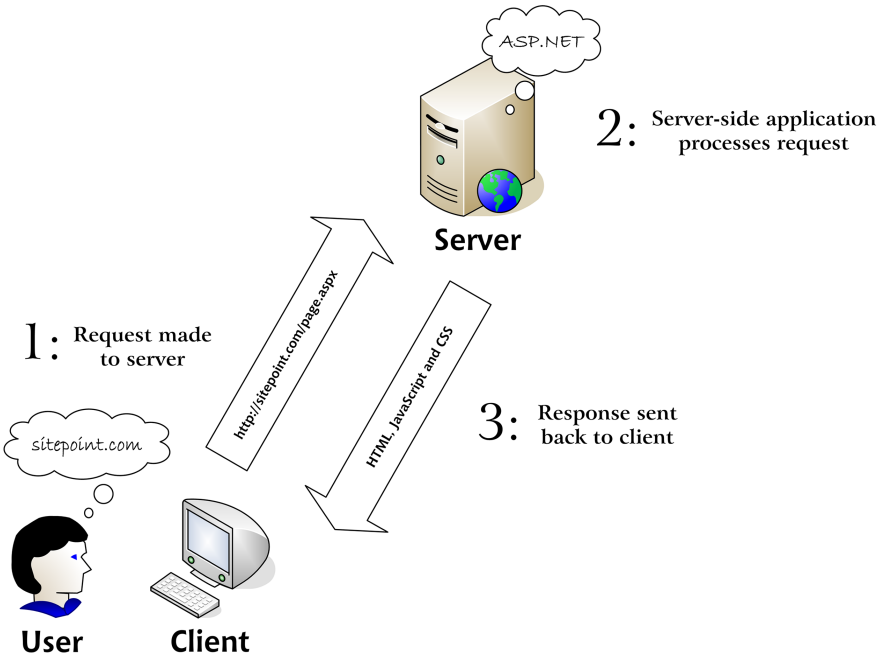

Figure 1.1. A user interacting with a web application

User The transaction starts and ends with the user. The user operates the web client software and interprets the results.

Web client This is the software program that the person uses to interact with the web application. The client is usually a web browser, such as Internet Explorer or Firefox.

Web server This is the software program located on the server. It processes requests made by the web client.

ASP.NET is a technology for developing web applications. A web application is just a fancy name for a dynamic website. A "website" can be thought of as a static page, where the content rarely changes or is purely informational only. Your local dentist or high school probably has a "website". A web application is dynamic in nature, and often considered to be a web version of standard desktop software. Google Mail is an excellent example of a web application. Web applications usually (but not al-

ways) store information in a database, and allow visitors to the site to access and change that information. Many different programming technologies and supported languages have been developed to create web applications; PHP, JSP, Ruby on Rails, CGI, and ColdFusion are just a few of the more popular ones. However, rather than tying you to a specific technology and language, ASP.NET lets you write web applications in a variety of familiar programming. We will focus only on the two most popular .NET languages, Visual Basic.NET (often referred to simply as VB.NET or VB) and C# (pronounced "See-Sharp").

ASP.NET uses the Microsoft .NET Framework. The .NET Framework collects all the technologies needed for building Windows desktop applications, web applications, web services, and so on into a single package, and makes them available to many programming languages. To say that ASP.NET uses the .NET Framework is really a huge understatement. ASP.NET is essentially the web version of what the .NET Framework is to the Windows desktop application world. For instance, if your friend wrote a really neat encryption library using .NET for a Windows desktop application, that code could be easily used within an ASP.NET web application with almost little to no changes.

Even with all the jargon explained, you're probably still wondering what makes ASP.NET so good. The truth is that there are many server-side technologies around, each of which has its own strengths and weaknesses. Yet ASP.NET has a few unique features:

- ASP.NET lets you write the server-side code using your favorite programming language— or at least the one you prefer from the long list of supported languages. The .NET Framework currently supports over 40 languages, and many of these may be used to build ASP.NET websites.

- ASP.NET pages are *compiled*, not interpreted. In ASP.NET's predecessor, ASP ("classic ASP"), pages were interpreted: every time a user requested a page, the server would read the page's code into memory, figure out how to execute the code, and execute it. In ASP.NET, the server need only figure out how to execute the code once. The code is compiled into efficient binary files, which can be run very quickly, again and again, without the overhead involved in rereading the page each time. This allows a big jump in performance, compared to the old days of ASP.

- ASP.NET has full access to the functionality of the .NET Framework. Support for XML, web services, database interaction, email, regular expressions, and many other technologies are built right into .NET, which saves you from having to reinvent the wheel.

- ASP.NET allows you to separate the server-side code in your pages from the HTML layout. When you're working with a team composed of programmers and design specialists, this separation is a great help, as it lets programmers modify the server-side code without stepping on the designers' carefully crafted HTML—and vice versa.

- ASP.NET makes it easy to reuse common User Interface elements in many web forms, as it allows us to save those components as independent web user controls. During the course of this book, you'll learn how to add powerful features to your website, and reuse them in many places with a minimum of effort.

- You can get excellent tools that assist in developing ASP.NET web applications. Visual Studio 2010 Express is a powerful, free visual editor that includes features such as a visual HTML editor, code autocompletion, code formatting, database integration functionality, debugging, and more. In the course of this book, you'll learn how to use this tool to build the examples we discuss.

- Security mechanisms such as membership roles and logins, as well as SQL Injection attack prevention, are automatically enabled out-of-the-box with an ASP.NET web app.

Still with us? Great! It's time to gather our tools and start building.

Installing the Required Software

If you're going to learn ASP.NET, you first need to make sure you have all the necessary software components installed and working on your system. Let's take care of this before we move on.

Visual Web Developer 2010 Express Edition

This is a powerful, free web development environment for ASP.NET 4.0. It includes features such as a powerful code, HTML and CSS editor, project debugging, IntelliSense (Microsoft's code autocompletion technology), database integration with the ability to design databases and data structures visually, and

much more. You're in for a lot of Visual Web Developer fun during the course of this book.

.NET Framework 4 and the .NET Framework Software Development Kit (SDK)
As we've already discussed, the .NET Framework drives ASP.NET. You're likely to have the .NET Framework already, as it installs automatically through the Windows Update service. Otherwise, it'll be installed together with Visual Studio.

Microsoft SQL Server 2008 R2 Express Edition
This is the free, but still fully functional, version of SQL Server 2008. This software is a Relational Database Management System whose purpose is to store, manage, and retrieve data as quickly and reliably as possible. You'll learn how to use SQL Server to store and manipulate the data for the DorkNozzle application you'll build in this book.

SQL Server Management Studio Express
Because the Express Edition of SQL Server doesn't ship with any visual management tools, you can use this free tool, also developed by Microsoft, to access your SQL Server 2008 database.

Installing Visual Web Developer 2010 Express Edition

Install Visual Web Developer 2010 Express Edition by following these simple steps:

1. Browse to http://www.microsoft.com/express/ and select **Microsoft Visual Studio**
2. Select the link for **Visual Web Developer 2010 Express** and click **Install Now**
3. On the Microsoft.com web page; click **Install Now**
4. .Execute the downloaded file, **vwd.exe**. This will begin the process for the Web Platform Installer.
5. As part of the installation of Visual Web Developer, you will install SQL Server 2008 R2 Express edition, which is identified as a dependency and automatically installed. The entire download is about 770MB.

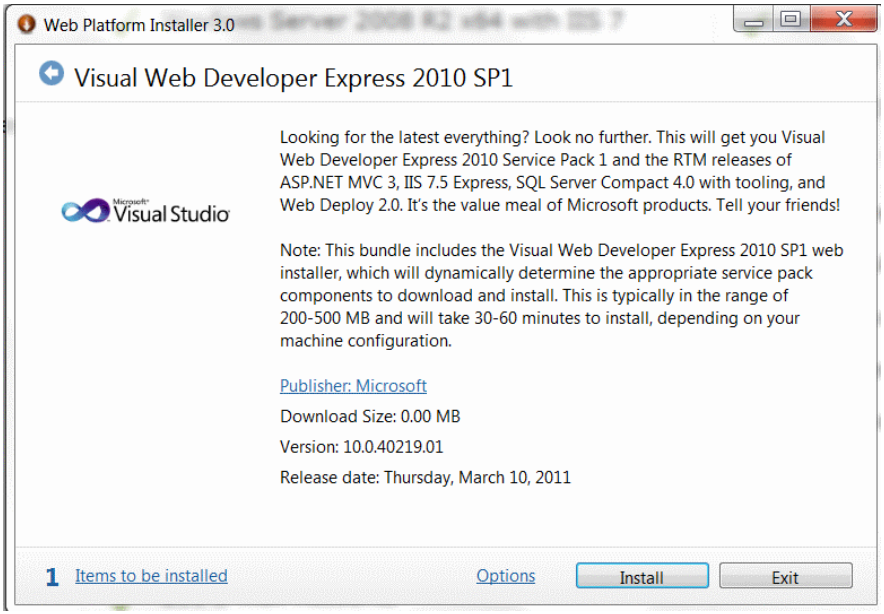

Figure 1.2. Installing Visual Web Developer 2010 Express Edition

6. In the next setup screen, you'll be asked to select the authentication mode for SQL Server 2008 R2 Express Edition. Here we choose to use Windows Authentication for simplicity going forward. Advanced users may choose to use mixed mode to set up their own account management with SQL Server, however, this book will assume the use of Windows Authentication mode.

7. The installer may prompt you to reboot your computer and possibly download more updates depending on your computer configuration. Please follow the on-screen instructions to ensure you have the latest versions.

8. Start Visual Web Developer to ensure it has installed correctly for you. Its welcome screen should look like Figure 1.3

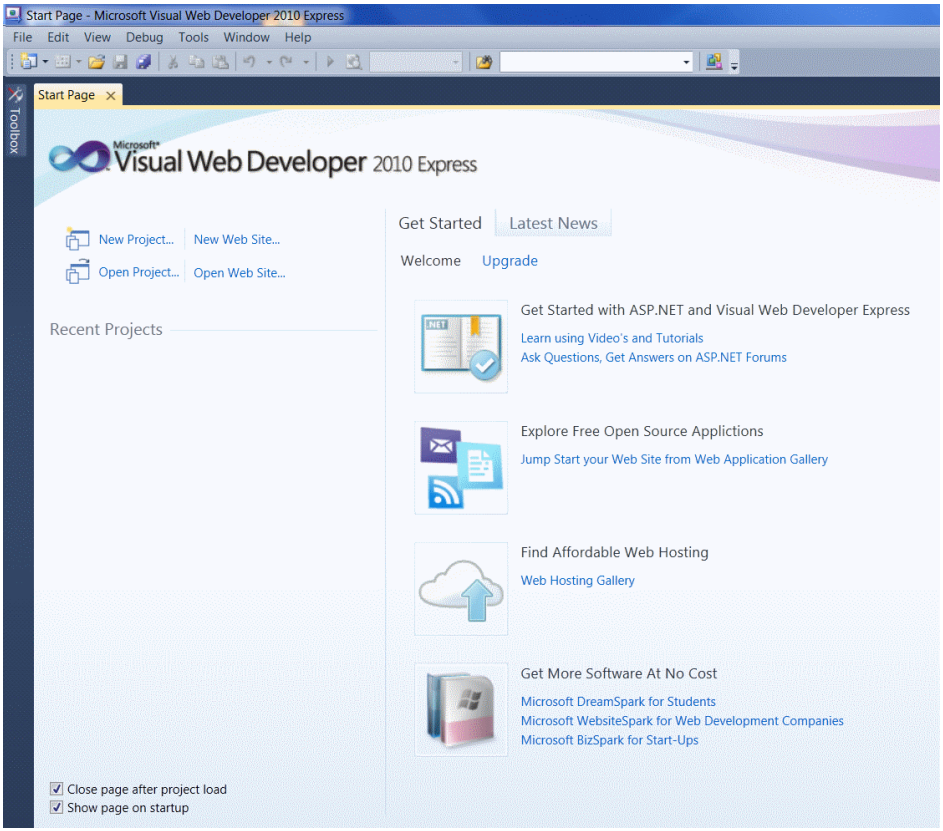

Figure 1.3. The start page of Visual Web Developer 2008 Express Edition

Installing SQL Server Management Studio Express

You've just installed Visual Web Developer and SQL Server 2008 R2 Express Editions. You won't use SQL Server until later in the book when we discuss relational databases, but we'll install all the required software here so that when the time comes, you'll have the complete environment set up.

In order to use your SQL Server 2008 instance effectively, you'll need an administration tool to work with your databases. SQL Server Management Studio Express is a free tool provided by Microsoft that allows you to manage your instance of SQL Server 2008. To install it, follow these steps:

1. Navigate to http://www.microsoft.com/express (or by using your favorite web search engine) and click the **Download** link under the **SQL Server Management Studio Express** section.

2. Download the file. After the download completes, execute the file and follow the steps to install the product. Be sure to choose the appropriate edition, whether 32-bit or 64-bit depending on your computer, with database tools. Be sure to choose to do a full install and under **Feature Selection** you check all boxes.

Once it's installed, SQL Server Manager Express can be accessed from **Start** > **All Programs** > **Microsoft SQL Server 2008** > **SQL Server Management Studio Express**. When executed, it will first ask for your credentials, asFigure 1.4 illustrates.

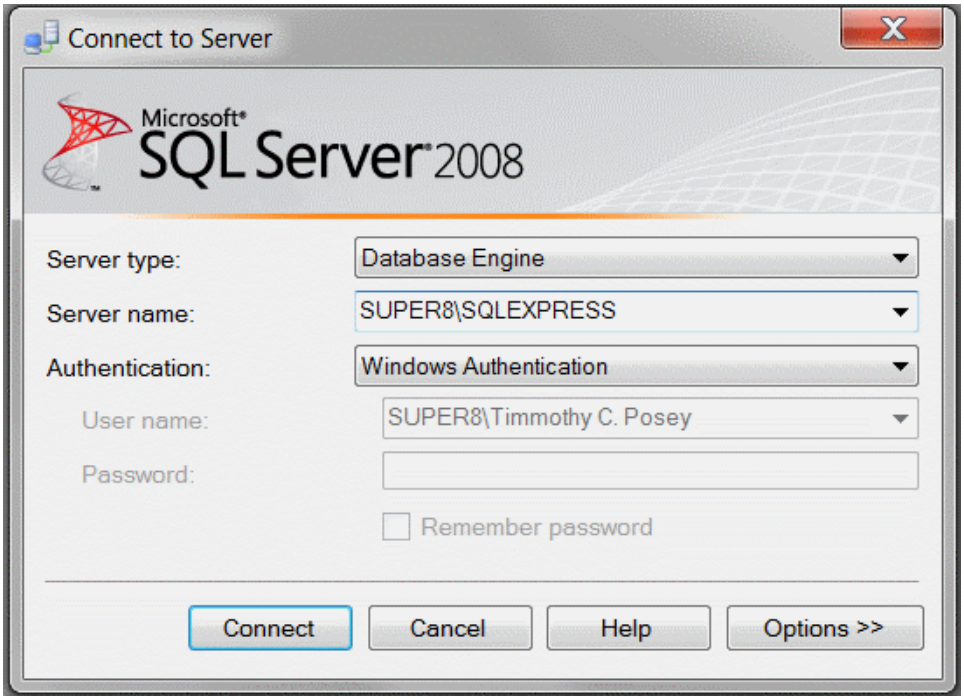

Figure 1.4. Connecting to SQL Server

By default, when installed, SQL Server 2008 Express Edition will only accept connections that use Windows Authentication, which means that you'll use your Windows user account to log into the SQL Server. Since you're the user that installed SQL Server 2008, you'll already have full privileges to the SQL Server. Click **Connect** to connect to your SQL Server 2008 instance.

After you're authenticated, you'll be shown the interface in Figure 1.5, which offers you many ways to interact with, and manage, your SQL Server 2008 instance.

SQL Server Management Studio lets you browse through the objects that reside on your SQL Server, and even modify their settings. For example, you can change the security settings of your server by right-clicking ***COMPUTER*\SQLEXPRESS** (where *COMPUTER* is the name of your computer), choosing **Properties**, and selecting **Security** from the panel, as shown in Figure 1.6. Here we've modified the **Server authentication** mode to `SQL Server and Windows Authentication mode`. We'll need this setting a bit later in the book, but you can set it now if you want, and then click **OK**.

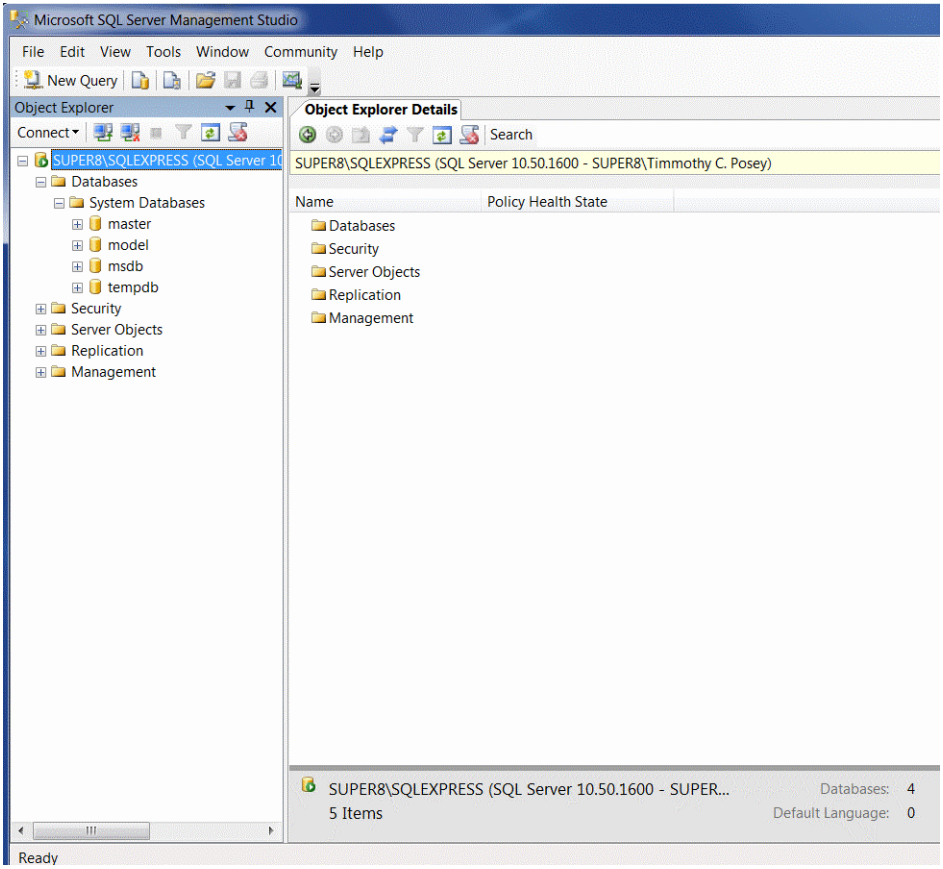

Figure 1.5. Managing your database server

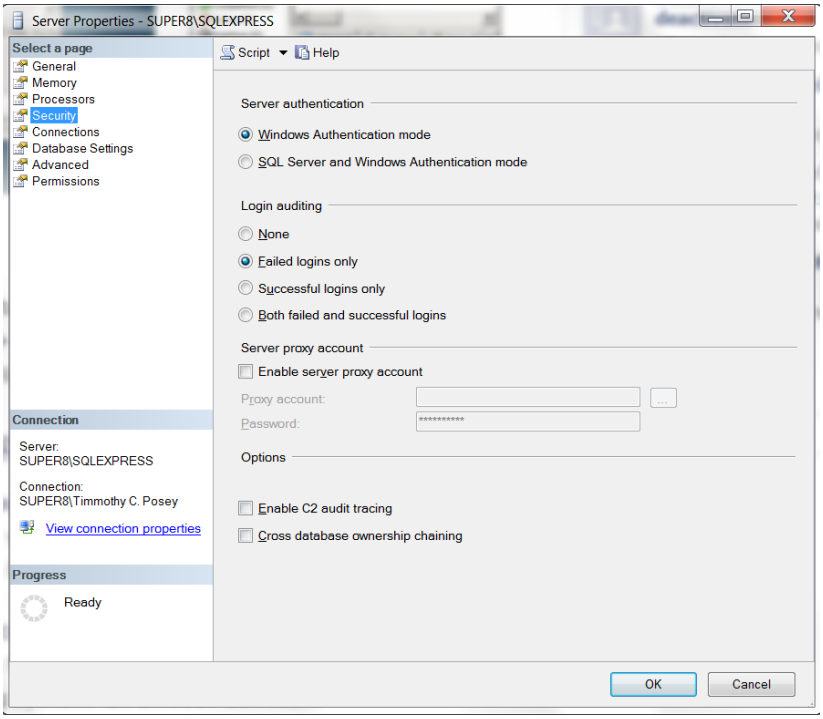

Figure 1.6. Changing server settings with SQL Server Management Studio

That's it. Your machine is now ready to build ASP.NET web projects and SQL Server databases. Now the fun starts—it's time to create your very first ASP.NET page!

Writing Your First ASP.NET Page

For your first ASP.NET exercise, we'll create the simple example shown in Figure 1.7. We'll go though the process of creating this page step by step.

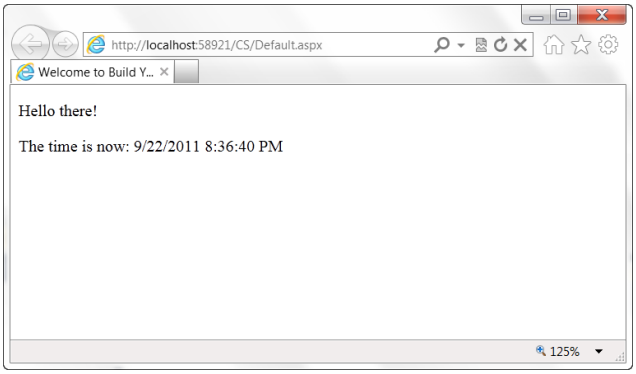

Figure 1.7. An exciting preview of your first ASP.NET page!

To create this page in Visual Web Developer, you'll need to follow a few simple steps:

1. Start Visual Web Developer, and choose **File** > **New Web Site** (or hit the default keyboard shortcut, **Shift+Alt+N**).

2. Choose `ASP.NET Web Site` for the template and `File System` for the location type. This location type tells Visual Web Developer to create the project in a physical folder on your disk, and execute that project using the integrated web server.

3. Choose the language in which you prefer to code your pages. Although ASP.NET allows you to code different pages inside a project in different languages, for the sake of simplicity we'll generally assume you work with a single language.

4. If you chose C# for the language, type **`C:\LearningASP\CS\`** for the folder location where you want to store the files for this exercise. If you prefer VB.NET, choose **`C:\LearningASP\VB\`**. You can choose any location you like. Figure 1.8 shows the C# version of the selection.

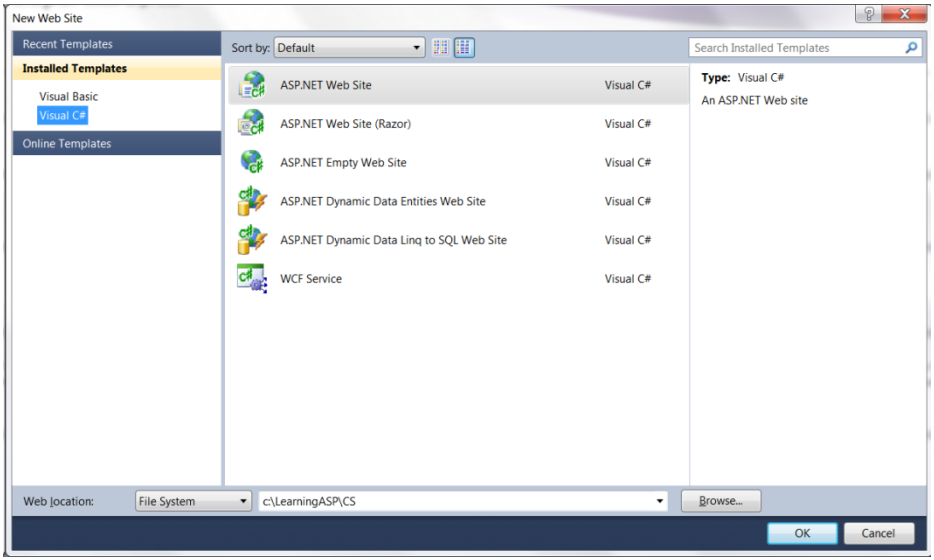

Figure 1.8. Starting a new ASP.NET Web Site project with Visual Web Developer

5. After clicking **OK**, Visual Web Developer will create the project along with several files to ease your transition into the ASP.NET development world. Your project will also come with a **Site.master** file, which represents a template applied to your entire site automatically. Your Project contains an empty **App_Data** folder, a **Scripts** folder which includes jQuery files, **Styles** which contains a basic **Site.css** stylesheet, a basic **Default.aspx** file, and a basic configuration file, **Web.config**—see Figure 1.9. We will discuss all of these files in Chapter 5, along with the purpose of the Account directory in detail. For now, let's jump right in to create our first ASP.NET web page.

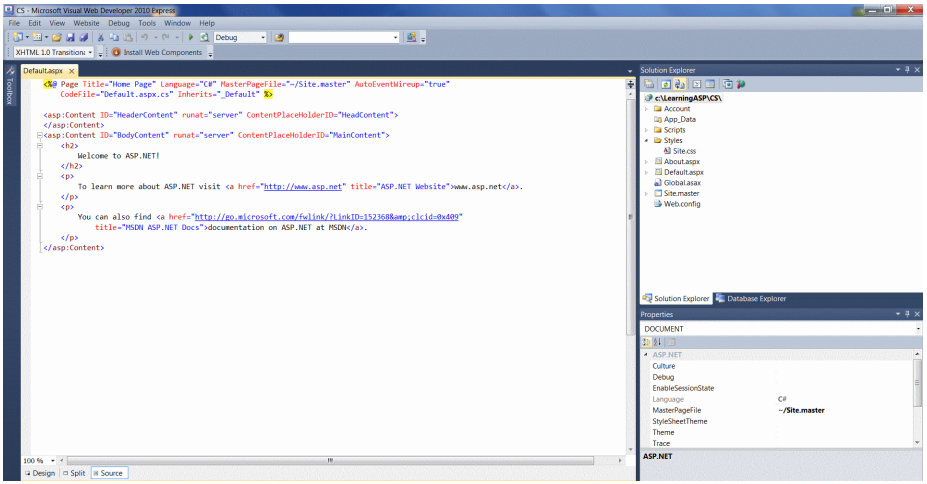

Figure 1.9. Your new project in Visual Web Developer

You may notice that the HTML source is different than standard HTML. This is normal. You should also notice that there are two content areas one for "Header Content" and one for "Main Content". Again, we will discuss templating and Master Pages in just a bit, but let's get immediately going. To do so, we can just overwrite the sample file provided to us.

The main panel in the Visual Web Developer interface is the page editor, in which you'll see the HTML source of the **Default.aspx** web page. Edit the title of the page to something more specific than `Home Page`, such as **`Welcome to Build Your Own ASP.NET 4 Website!`**:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>Welcome to Build Your Own ASP.NET 4 Website!
    </title>
  </head>
```

Loading the **Default.aspx** page in a web browser now opens the sample page that was created when Visual Web Developer created the project; this makes sense, since we didn't add any content to this page! Because we don't care for any of the sample page, we will simply modify the entire source code for the **default.aspx** page as follows:

```
  <body>
    <form id="form1" runat="server">
    <div>
      <p>Hello there!</p>
      <p>
        The time is now:
        <asp:Label ID="myTimeLabel" runat="server" />
      </p>
    </div>
    </form>
  </body>
</html>
```

Although our little page isn't yet finished (our work with the Label control isn't over), let's execute the page to ensure we're on the right track. Hit **F5** or go to **Debug** menu.

How a Web Server Control Works

You've just added a Web Server Control to the page by adding an `<asp:Label/>` element to the page. You'll learn all about Web Server Controls in Chapter 2, but for now you need to learn how this simple control works so that you can understand the exercise.

The `Label` control is one of the simplest controls in .NET, which lets you insert dynamic content into the page. The `asp:` part of the tag name identifies it as a built-in ASP.NET tag. ASP.NET comes with numerous built-in tags, and `<asp:Label/>` is probably one of the most frequently used.

The `runat="server"` attribute value identifies the tag as something that needs to be handled on the server. In other words, the web browser will never see the `<asp:Label/>` tag; when the page is requested by the client, ASP.NET sees it and converts it to regular HTML tags *before* the page is sent to the browser. It's up to us to write the code that will tell ASP.NET to replace this particular tag with something meaningful to the user loading the page.

The first time you do this, Visual Web Developer will let you know that your project isn't configured for debugging, and it'll offer to make the necessary change to the configuration (**Web.config**) file for you—see Figure 1.10. Confirm the change by clicking **OK**.

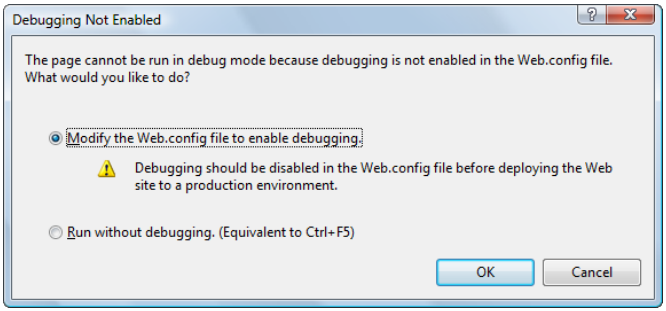

Figure 1.10. Enabling project debugging in Visual Web Developer

If `Script Debugging` is not enabled in Internet Explorer, you'll get the dialog shown in Figure 1.11. Check the **Don't show this dialog again** checkbox, and click **Yes**.

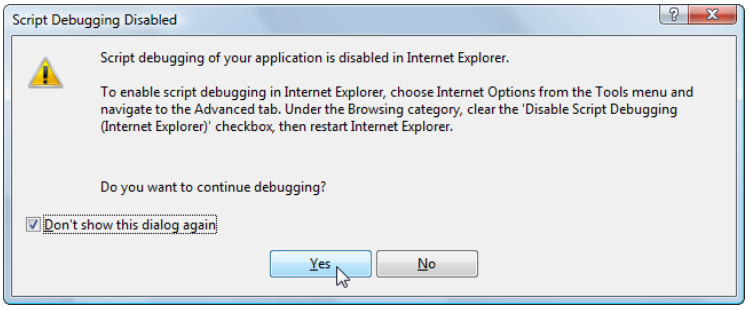

Figure 1.11. Enabling script debugging in Internet Explorer

After all the notifications are out of the way, you should have a page like that in Figure 1.12:

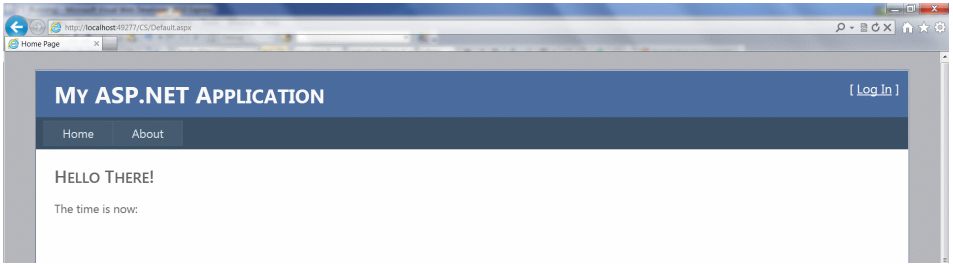

Figure 1.12. Executing your first ASP.NET web page

You can now close the Internet Explorer window. Visual Web Developer will automatically detect this action and will cancel debugging mode, allowing you to start editing the project again. Now let's do something with that Label control.

Set Your Default Browser to Internet Explorer

When executing the project, the website is loaded in your system's default web browser. For the purposes of developing ASP.NET applications, we recommend configuring Visual Web Developer to use Internet Explorer, even if this is not your preferred web browser. We recommend Internet Explorer because it integrates better with Visual Web Developer's .NET and JavaScript debugging features. For example, Visual Web Developer knows to automatically stop debugging the project when the Internet Explorer window is closed. To change the default browser to be used by Visual Web Developer, right-click the root node in **Solution Explorer**, choose **Browse With**, select a browser from the **Browsers** tab, and click **Set as Default**.

For our first dynamic web page using ASP.NET, let's write some code that will display the current time inside the `Label` control. That mightn't sound very exciting, but it's only for the purposes of this simple demonstration; don't worry, we'll reach the good stuff before too long. To programmatically manipulate the `Label` control, you'll have to write some C# or VB.NET code, depending on the language you've chosen when creating the project. As suggested earlier in this chapter, ASP.NET allows **web forms** (**.aspx** pages) to contain C# or VB.NET code, or they can use separate files—named **code-behind files**—for storing this code. The **Default.aspx** file that was generated for you when creating the project was generated with a code-behind file, and we want to edit that file now. There are many ways in which you can open that file. You can click the **View Code** icon at the top of the **Solution Explorer** window, right-click the **Default.aspx** file in **Solution Explorer** and choose **View Code**, or click the + symbol to expand the **Default.aspx** entry. No matter how you open this file, it should look like Figure 1.13 if you're using C#, or Figure 1.14 if you're using VB.NET.

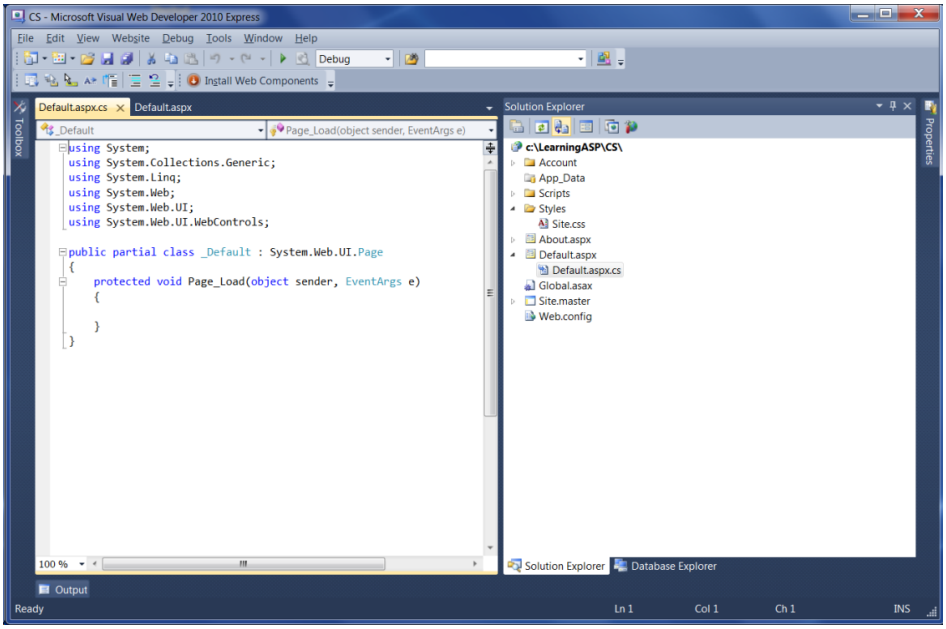

Figure 1.13. Default.aspx.cs in Visual Web Developer

C#, VB.NET, and Visual Web Developer

You may be slightly alarmed, at first, by the fact that the code-behind file template that Visual Web Developer generates for the **Default.aspx** file in a new project when you're using C# is completely different from the one it generates when you're using VB.NET. They're based on the same platform and use the same data types and features, so C# and VB.NET are fundamentally very similar. However, there are still large differences between the languages' syntax. VB.NET is frequently preferred by beginners because its syntax is perceived to be easier to read and understand than C#. While the differences can be intimidating initially, after we discuss their details in Chapter 3, you'll see that it can be relatively easy to understand both.

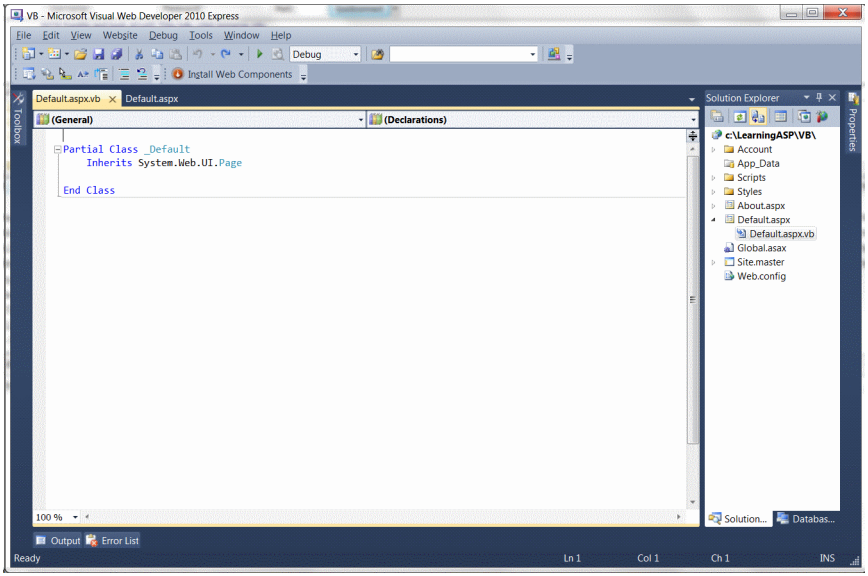

Figure 1.14. Default.aspx.vb in Visual Web Developer

Looking at Figure 1.13 and Figure 1.14 you can see that the C# version contains a definition for a method called `Page_Load`, while the VB.NET version doesn't. This is the method that executes automatically when the project is executed, and we want to use it to write the code that will display the current time inside the `Label` control.

If you're using VB.NET, you'll need to generate the `Page_Load` method first. The easiest way to have Visual Web Developer generate `Page_Load` for you is to open **Default.aspx**—*not* its code-behind file—and switch to **Design** view (as shown in Figure 1.15). If you double-click on an empty place on the form, an empty `Page_Load` method will be created in the code-behind file for **Default.aspx**.

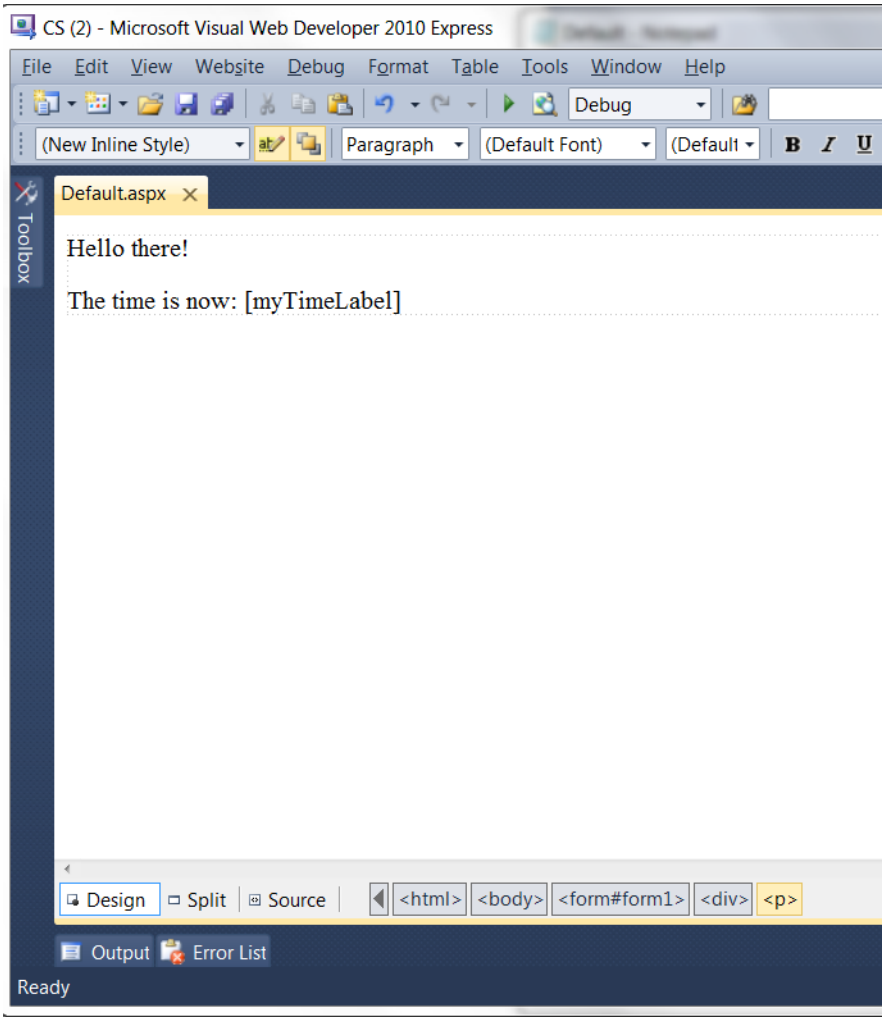

Figure 1.15. **Default.aspx** in **Design** view in Visual Web Developer

Now edit the `Page_Load` method so that it looks like this, selecting the version that applies to your chosen language:

Visual Basic — LearningASP\VB\Default.aspx.vb *(excerpt)*

```
Partial Class _Default
  Inherits System.Web.UI.Page
  Protected Sub Page_Load(ByVal sender As Object,
  ➥  ByVal e As System.EventArgs)
      Handles Me.Load
```

```
    myTimeLabel.Text = DateTime.Now.ToString()
  End Sub
End Class
```

C# LearningASP\CS\Default.aspx.cs *(excerpt)*

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    myTimeLabel.Text = DateTime.Now.ToString();
  }
}
```

C# is Case Sensitive

C#, unlike VB, is case sensitive. If you type the case of a letter incorrectly, the page won't load. If these languages look complicated, don't worry: you'll learn more about them in Chapter 3.

If you've never done any server-side programming before, the code may look a little scary. But before we analyze it in detail, let's load the page and test that it works for real. To see your dynamically generated web page content in all its glory, hit **F5** to execute the project again, and see the current date and time, as depicted in Figure 1.16.

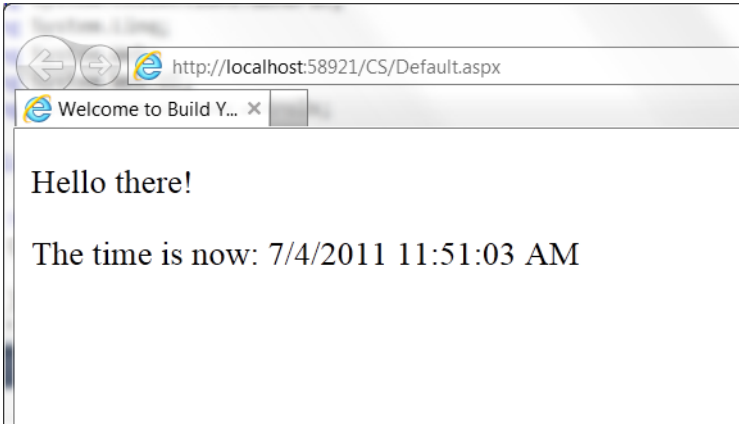

Figure 1.16. Loading **Default.aspx** with a `Label` element with dynamically generated content

Both versions of the page achieve exactly the same thing. You can even save them both, giving each a different filename, and test them separately. Alternatively, you can create two Visual Web Developer projects—one for C# code, in **C:\LearningASP\CS**, and one for VB.NET code, in **C:\LearningASP\VB**.

No Time?

If the time isn't displayed in the page, chances are that you opened the file directly in your browser instead of loading it through your web server. Because ASP.NET is a server-side language, your web server needs to process the file before it's sent to your browser for display. If it doesn't gain access to the file, the ASP.NET code is never converted into HTML that your browser can understand, so make sure you load the page by executing it in Visual Web Developer. Loading the page in your browser directly from Windows Explorer will not execute the code, and consequently the time won't display.

So how does the code work? Let's break down some of the elements that make up the page. We're defining a method called `Page_Load`, in both languages:

Visual Basic — LearningASP\VB\Default.aspx.vb *(excerpt)*

```
Protected Sub Page_Load(ByVal sender As Object,
➥  ByVal e As System.EventArgs)
    Handles Me.Load
```

C# — LearningASP\CS\Default.aspx.cs *(excerpt)*

```
protected void Page_Load(object sender, EventArgs e)
{
```

I won't go into too much detail here. For now, all you need to know is that you can write script fragments that are run in response to different events, such as a button being clicked or an item being selected from a drop-down. What the first line of code basically says is, "execute the following script whenever the page is loaded." Note that C# groups code into blocks with curly braces ({ and }), while Visual Basic uses statements such as `End Sub` to mark the end of a particular code sequence. So, the curly brace ({) in the C# code above marks the start of the script that will be executed when the page loads for the first time.

Here's the line that actually displays the time on the page:

Visual Basic LearningASP\VB\Default.aspx.vb *(excerpt)*

```
    myTimeLabel.Text = DateTime.Now.ToString()
```

C# LearningASP\CS\Default.aspx.cs *(excerpt)*

```
    myTimeLabel.Text = DateTime.Now.ToString();
```

As you can see, these .NET languages have much in common, because they're both built on the .NET Framework. In fact, the only difference between the ways the two languages handle the above line is that C# ends lines of code with a semicolon (`;`). In plain English, here's what this line says:

```
Set the Text of myTimeLabel to the current date and time, expressed
as text
```

Note that `myTimeLabel` is the value we gave for the `id` attribute of the `<asp:Label/>` tag where we want to show the time. So, `myTimeLabel.Text`, or the `Text` property of `myTimeLabel`, refers to the text that will be displayed by the tag. `DateTime` is a class that's built into the .NET Framework; it lets you perform all sorts of useful functions with dates and times. The .NET Framework has thousands of these classes, which do countless handy things. The classes are collectively known as the **.NET Framework Class Library**.

The `DateTime` class has a property called `Now`, which returns the current date and time. This `Now` property has a method called `ToString`, which expresses that date and time as text (a segment of text is called a **string** in programming circles). Classes, properties, and methods: these are all important words in the vocabulary of any programmer, and we'll discuss them in more detail a little later in the book. For now, all you need to take away from this discussion is that `DateTime.Now.ToString()` will give you the current date and time as a text string, which you can then tell your `<asp:Label/>` tag to display.

The rest of the script block simply ties up loose ends. The End Sub in the VB code, and the } in the C# code, mark the end of the script that's to be run when the page is loaded:

Visual Basic LearningASP\VB\Default.aspx.vb *(excerpt)*

```
  End Sub
```

C# LearningASP\CS\Default.aspx.cs *(excerpt)*

```
  }
```

One final thing that's worth investigating is the code that ASP.NET generated for you. It's clear by now that your web browser receives only HTML (no server-side code!), so what kind of HTML was generated for that label? The answer is easy to find! With the page displayed in your browser, you can use the browser's **View Source** feature to view the page's HTML code. In the middle of the source, you'll see something like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
      Welcome to Build Your Own ASP.NET 4 Web Site!
    </title>
  </head>
  <body>
    <form name="form1" method="post" action="Default.aspx"
        id="form1">
    <div>
      <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
          value="…" />
    </div>
    <div>
      <p>Hello there!</p>
      <p>
        The time is now:
        <span id="myTimeLabel">5/13/2008 3:10:38 PM</span>
      </p>
    </div>
    </form>
  </body>
</html>
```

Notice that all the ASP.NET code has gone? Even the `<asp:Label/>` tag has been replaced by a `<span>` tag (which has the same id attribute as the `<asp:Label/>` tag we used) that contains the date and time. There's a mysterious hidden `input` element named `__VIEWSTATE` that is used by ASP.NET for certain purposes, but we'll ignore it for now. (Don't worry, we'll discuss it a bit later in the book!)

That's how ASP.NET works. From the web browser's point of view, there's nothing special about an ASP.NET page; it's just plain HTML like any other. All the ASP.NET code is run by your web server and converted to plain HTML that's sent to the browser. So far, so good, but the example above was fairly simple. The next chapter will be a bit more challenging as we investigate some valuable programming concepts.

Getting Help

As you develop ASP.NET web applications, you'll undoubtedly have questions that need answers, and problems that need to be solved. Help is at hand—Microsoft has developed the ASP.NET support website.[1] This portal provides useful information for the ASP.NET community, such as news, downloads, articles, and discussion forums. You can also ask questions of the experienced community members in the SitePoint Forums.[2]

Summary

In this chapter, you learned about .NET, including the benefits of ASP.NET, and that it's a part of the .NET Framework.

First, we covered the components of ASP.NET. Then we explored the software that's required not only to use this book, but also in order to progress with ASP.NET development.

You've gained a solid foundation in the basics of ASP.NET. The next chapter will build on this knowledge as we begin to introduce you to ASP.NET in more detail, covering page structure, the languages that you can use, various programming concepts, and the finer points of form processing.

[1] http://www.asp.net/
[2] http://www.sitepoint.com/forums/

Chapter 2

ASP.NET Basics

So far, you've learned what ASP.NET is, and what it can do. You've installed the software you need to get going, and you even know how to create a simple ASP.NET page. Don't worry if it all seems a little bewildering right now: as this book progresses, you'll learn how easy it is to use ASP.NET at more advanced levels.

As the next few chapters unfold, we'll explore some more advanced topics, including the use of controls and various programming techniques. But before you can begin to develop applications with ASP.NET, you'll need to understand the inner workings of a typical ASP.NET page—with this knowledge, you'll be able to identify the parts of the ASP.NET page referenced in the examples we'll discuss throughout this book. So in this chapter, we'll talk about some key mechanisms of an ASP.NET page, specifically:

- page structure
- view state
- namespaces
- directives

We'll also cover more of VB and C#, two of the "built-in" languages supported by the .NET Framework: VB and C#. As this section progresses, we'll explore the differences and similarities between them, and form a clear idea of the power that they provide for those creating ASP.NET applications.

So, what exactly makes up an ASP.NET page? The next few sections will give you an in-depth understanding of the constructs of a typical ASP.NET page.

ASP.NET Page Structure

ASP.NET pages are simply text files that have the **.aspx** file name extension, and can be placed on any web server equipped with ASP.NET.

When a client requests an ASP.NET page, the web server passes the page to the **ASP.NET runtime**, a program that runs on the web server that's responsible for reading the page and compiling it into a .NET class. This class is then used to produce the HTML that's sent back to the user. Each subsequent request for this page avoids the compilation process: the .NET class can respond directly to the request, producing the page's HTML and sending it to the client until such time as the **.aspx** file changes. This process is illustrated in Figure 2.1.

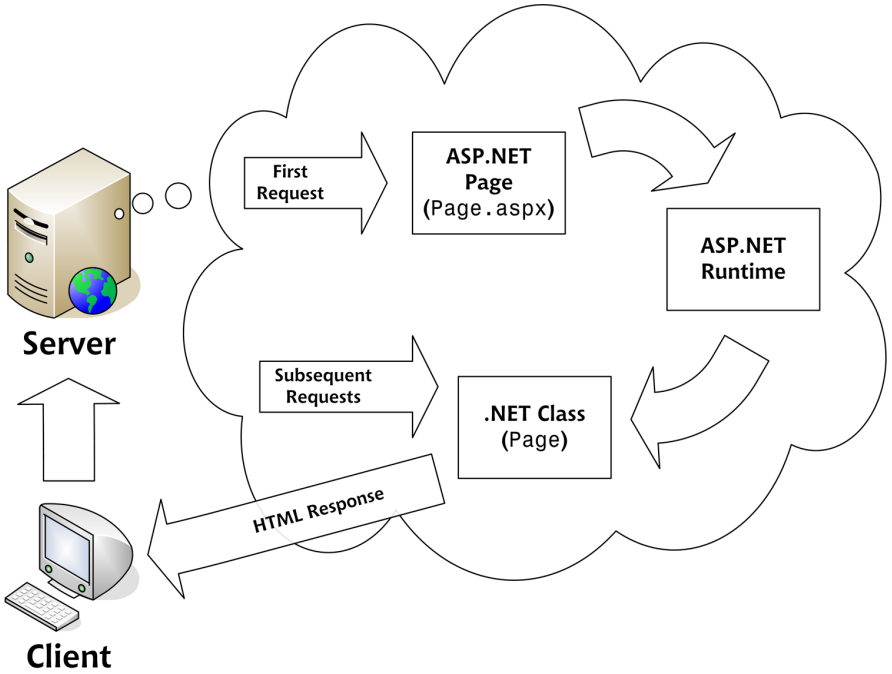

Figure 2.1. The life cycle of the ASP.NET page

An ASP.NET page consists of the following elements:

- directives
- code declaration blocks
- code render blocks
- ASP.NET server controls
- server-side comments
- literal text and HTML tags

For the purpose of examining all the elements that can make up an ASP.NET page, we will not be using any code-behind files as we did in Chapter 1. Code-behind files are useful for separating layout from code by breaking a web form into two files, but here all we're interested in seeing is all the pieces of a web form in one place. This will make it easier to understand the structure of the web form.

The code below represents a version of the page you wrote in Chapter 1, which does not use a code-behind file. You'll notice the server-side script code now resides in a `script` element:

Visual Basic — LearningASP\VB\Hello.aspx *(excerpt)*

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  Protected Sub Page_Load(ByVal sender As Object,
  ➥  ByVal e As System.EventArgs)
    myTimeLabel.Text = DateTime.Now.ToString()
  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>Welcome to Build Your Own ASP.NET 4 Web Site!</title>
  </head>
  <body>
    <form id="form1" runat="server">
    <div>
      <p>Hello there!</p>
      <p>
        The time is now:
        <%-- Display the current date and time --%>
        <asp:Label ID="myTimeLabel" runat="server" />
      </p>
      <p>
        <%-- Declare the title as string and set it --%>
        <% Dim Title As String = "This is generated by a code
            render block."%>
        <%= Title %>
      </p>
    </div>
    </form>
  </body>
</html>
```

C# — LearningASP\CS\Hello.aspx *(excerpt)*

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {
    myTimeLabel.Text = DateTime.Now.ToString();
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>Welcome to Build Your Own ASP.NET 4 Web Site!</title>
  </head>
  <body>
    <form id="form1" runat="server">
    <div>
      <p>Hello there!</p>
      <p>
        The time is now:
        <%-- Display the current date and time --%>
        <asp:Label ID="myTimeLabel" runat="server" />
      </p>
      <p>
        <%-- Declare the title as string and set it --%>
        <% string Title = "This is generated by a code render
            block."; %>
        <%= Title %>
      </p>
    </div>
    </form>
  </body>
</html>
```

If you like, you can save this piece of code in a file named **Hello.aspx** within the **LearningASP\CS** or **LearningASP\VB** directory you created in Chapter 1. You can open a new file in Visual Web Developer by selecting **Website** > **Add New Item...**. Use a **Web Form** template, and, since we are not using a code-behind file for this example, you need to deselect the **Place code in a separate file** checkbox. Alternatively, you can copy the file from the source code archive.

Executing the file (by hitting **F5**) should render the result shown in Figure 2.2.

Figure 2.2. Sample page in action

This ASP.NET page contains examples of all the above components (except server-side includes) that make up an ASP.NET page. You won't often use every single element in a given page, but it's important that you're familiar with these elements, their purposes, and how and when it's appropriate to use them.

Directives

The directives section is one of the most important parts of an ASP.NET page. Directives control how a page is compiled, specify how a page is cached by web browsers, aid debugging (error-fixing), and allow you to import classes to use within your page's code. Each directive starts with `<%@`. This is followed by the directive name, plus any attributes and their corresponding values. The directive then ends with `%>`.

There are many directives that you can use within your pages, and we'll discuss them in greater detail later. For the moment, however, know that the `Import` and `Page` directives are the most useful for ASP.NET development. Looking at our sample ASP.NET page, **Hello.aspx**, we can see that a `Page` directive was used at the top of the page like so:

Visual Basic LearningASP\VB\Hello.aspx *(excerpt)*

```
<%@ Page Language="VB" %>
```

C# LearningASP\CS\Hello.aspx *(excerpt)*

```
<%@ Page Language="C#" %>
```

In this case, the `Page` directive specifies the language that's to be used for the application logic by setting the `Language` attribute. The value provided for this attribute, which appears in quotes, specifies that we're using either VB or C#. A whole range of different directives is available; we'll see a few more later in this chapter.

ASP.NET directives can appear anywhere on a page, but they're commonly included at its very beginning.

Code Declaration Blocks

In Chapter 3, we'll talk more about code-behind pages and how they let us separate our application logic from an ASP.NET page's HTML. However, if you're not working with code-behind pages, you must use code declaration blocks to contain all the application logic of your ASP.NET page. This application logic defines variables, subroutines, functions, and more. In our sample page, we've placed the code inside `<script>` tags with the `runat="server"` attribute, like so:

Visual Basic LearningASP\VB\Hello.aspx *(excerpt)*

```
<script runat="server">
  Protected Sub Page_Load(ByVal sender As Object,
  ➥  ByVal e As System.EventArgs)
    'set the label text to the current time
    myTimeLabel.Text = DateTime.Now.ToString()
  End Sub
</script>
```

C# LearningASP\CS\Hello.aspx *(excerpt)*

```
<script runat="server">
  protected void Page_Load(object sender, EventArgs e)
  {
    //set the label text to the current time
```

```
    myTimeLabel.Text = DateTime.Now.ToString();
  }
</script>
```

Comments in VB and C# Code

Both of these code snippets contain comments—explanatory text that will be ignored by ASP.NET, but which serves to describe to us how the code works.

In VB code, a single quote or apostrophe (`'`) indicates that the remainder of the line is to be ignored as a comment, while in C# code, two slashes (`//`) achieve the same end:

Visual Basic — LearningASP\VB\Hello.aspx *(excerpt)*

```
    'set the label text to the current time
```

C# — LearningASP\CS\Hello.aspx *(excerpt)*

```
    //set the label text to the current time
```

C# code also lets us span a comment over multiple lines if we begin it with `/*` and end it with `*/`, as in this example:

```
/*set the label text
  to the current time */
```

<script> Tag Attributes

Before .NET emerged, ASP also supported such script tags using a `runat="server"` attribute. However, they could only ever contain VBScript and, for a variety of reasons, they failed to find favor among developers.

The `<script runat="server">` tag accepts two other attributes: `language` and . We can set the language that's used in this code declaration block via the `language` attribute:

Visual Basic

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

C#

```
<script runat="server" language="C#">
```

If you don't specify a language within the code declaration block, the ASP.NET page will use the language provided by the `language` attribute of the `Page` directive. Each page's code must be written in a single language; for instance, it's not possible to mix VB and C# in the same page.

`src`The second attribute that's available to us is `src`; this lets us specify an external code file for use within the ASP.NET page:

Visual Basic

```
<script runat="server" language="VB" src="mycodefile.vb">
```

C#

```
<script runat="server" language="C#" src="mycodefile.cs">
```

We also can use these `<script>` blocks to write JavaScript code. This is especially useful for making our web pages more dynamic.

Code Render Blocks

You can use code render blocks to define inline code or expressions that will execute when a page is rendered. Code within a code render block is executed immediately when it is encountered during page rendering. On the other hand, code within a code *declaration* block (within `<script>` tags) is executed only when it is called or triggered by user or page interactions. There are two types of code render blocks—inline code, and inline expressions—both of which are typically written within the body of the ASP.NET page.

Inline code render blocks execute one or more statements, and are placed directly inside a page's HTML between `<%` and `%>` delimiters. In our example, the following is a code render block:

Visual Basic LearningASP\VB\Hello.aspx *(excerpt)*

```
<% Dim Title As String = "This is generated by a code render
➥  block." %>
```

C# LearningASP\CS\Hello.aspx *(excerpt)*

```
<% string Title = "This is generated by a code render block."; %>
```

These code blocks simply declare a String variable called `Title`, and assign it the value `This is generated by a code render block`.

Inline expression render blocks can be compared to `Response.Write` in classic ASP. They start with `<%=` and end with `%>`, and are used to display the values of variables and the results of methods on a page. In our example, an inline expression appears immediately after our inline code block:

Visual Basic LearningASP\VB\Hello.aspx *(excerpt)*

```
<%= Title %>
```

C# LearningASP\CS\Hello.aspx *(excerpt)*

```
<%= Title %>
```

If you're familiar with classic ASP, you'll know what this code does: it simply outputs the value of the variable `Title` that we declared in the previous inline code block.

As a new feature of the ASP.NET 4 release, you can also use code render blocks to "HTML Encode" your output. Html encoding is a fancy term for ensuring your text is output so it can be safely read by the browser instead of having it interpreted as HTML code itself. For instance, if you wanted to put a less-than < or greater-than sign >, these would automatically be interpreted by the browser as HTML tags. Html encoding ensures the actual characters are displayed. Consider the following example:

```
<%: "5 is > 3" %>
```

ASP.NET Server Controls

At the heart of any ASP.NET page lie server controls, which represent dynamic elements with which your users can interact. There are three basic types of server control: ASP.NET controls, HTML controls, and web user controls.

Usually, an ASP.NET control must reside within a `<form runat="server">` tag in order to function correctly. Controls offer the following advantages to ASP.NET developers:

- They give us the ability to access HTML elements easily from within our code: we can change these elements' characteristics, check their values, or even update them dynamically from our server-side programming language of choice.
- ASP.NET controls retain their properties thanks to a mechanism called **view state**. We'll be covering view state later in this chapter. For now, you need to know that view state prevents users from losing the data they've entered into a form once that form has been sent to the server for processing. When the response comes back to the client, text box entries, drop-down list selections, and so on are all retained through view state.
- With ASP.NET controls, developers are able to separate a page's presentational elements (everything the user sees) from its application logic (the dynamic portions of the ASP.NET page), so that each can be maintained separately.
- Many ASP.NET controls can be "bound" to the data sources from which they will extract data for display with minimal (if any) coding effort.

ASP.NET is all about controls, so we'll be discussing them in greater detail as we move through this book. In particular, Chapter 4 explains many of the controls that ship with ASP.NET. For now, though, let's continue our dissection of an ASP.NET page.

Server-side Comments

Server-side comments allow you to include within the page comments or notes that won't be processed by ASP.NET. Traditional HTML uses the `<!--` and `-->` character sequences to delimit comments; any information included between these tags won't be displayed to the user. ASP.NET comments look very similar, but use the sequences `<%--` and `--%>`.

Our ASP.NET example contains two server-side comment blocks, the first of which is:

LearningASP\VB\Hello.aspx *(excerpt)*

```
<%-- Display the current date and time --%>
```

The difference between ASP.NET comments and HTML comments is that ASP.NET comments are not sent to the client at all; HTML comments are, so they're not suited to commenting out ASP.NET code. Consider the following example:

C#

```
<!--
<% string Title = "This is generated by a code render block."; %>
<%= Title %>
-->
```

Here, it looks as though a developer has attempted to use an HTML comment to stop a code render block from being executed. Unfortunately, HTML comments will only hide information from the browser, not the ASP.NET runtime. So in this case, although we won't see anything in the browser that represents these two lines, they will be processed by ASP.NET, and the value of the variable `Title` will be sent to the browser inside an HTML comment, as shown here:

```
<!--

This is generated by a code render block.
-->
```

The code could be modified to use server-side comments very simply:

C#

```
<%--
<% string Title = "This is generated by a code render block."; %>
<%= Title %>
--%>
```

The ASP.NET runtime will ignore the contents of this comment, and the value of the `Title` variable will not be output.

Literal Text and HTML Tags

The final elements of an ASP.NET page are plain old text and HTML. Generally, you can't do without these elements—after all, HTML allows the display of the information in your ASP.NET controls and code in a way that's suitable for users and their browsers. Let's take a look at the literal text and HTML tags that were used to produce the display in the Visual Basic version of our sample page (the text and HTML in the C# version is identical):

Visual Basic — LearningASP\VB\Hello.aspx *(excerpt)*

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  Protected Sub Page_Load(ByVal sender As Object,
  ➥  ByVal e As System.EventArgs)
    myTimeLabel.Text = DateTime.Now.ToString()
  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>Welcome to Build Your Own ASP.NET 4 Web Site!</title>
  </head>
  <body>
    <form id="form1" runat="server">
    <div>
      <p>Hello there!</p>
      <p>
        The time is now:
        <%-- Display the current date and time --%>
        <asp:Label ID="myTimeLabel" runat="server" />
      </p>
      <p>
        <%-- Declare the title as string and set it --%>
        <% Dim Title As String = "This is generated by a code
            render block."%>
        <%= Title %>
```

```
      </p>
    </div>
    </form>
  </body>
</html>
```

The bold code above highlights the fact that literal text and HTML tags provide the structure for presenting our dynamic data. Without these elements, this page would have no format, and the browser would be unable to understand it.

By now, you should have a clearer understanding of the structure of an ASP.NET page. As you work through the examples in this book, you'll begin to realize that, in many cases, you won't need to use all of these elements. For the most part, your development will be modularized within code-behind files or code declaration blocks, and all of the dynamic portions of your pages will be contained within code render blocks or controls located inside a `<form runat="server">` tag.

In the following sections, we'll explore view state, discuss working with directives, and shine a little light on the languages that can be used within ASP.NET.

View State

ASP.NET controls automatically retain their data when a page is sent to the server in response to an event (such as a user clicking a button). Microsoft calls this persistence of data **view state**. In the past, developers would've had to resort to hacks to have the application remember the item a user had selected in a drop-down menu, or store the content entered into a text box; typically, these hacks would have relied on hidden form fields.

This is no longer the case. Once they're submitted to the server for processing, ASP.NET pages automatically retain all the information contained in text boxes and drop-down lists, as well as radio button and checkbox selections. They even keep track of dynamically generated tags, controls, and text. Consider the following code written in the "ancient" ASP (not ASP.NET!) framework:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
```

```
    <title>Sample Page using VBScript</title>
  </head>
  <body>
    <form method="post" action="sample.asp">
      <input type="text" name="nameTextBox"/>
      <input type="submit" name="submitButton"
          value="Click Me" />
      <%
      If Request.Form("nameTextBox") <> "" Then
        Response.Write(Request.Form("nameTextBox"))
      End If
      %>
    </form>
  </body>
</html>
```

Loading this page through an ASP-enabled web server (such as IIS) would reveal that the view state is not automatically preserved. When the user submits the form, the information that was typed into the text box is cleared, although it's still available in the `Request.Form("nameTextBox")` property. The equivalent page in ASP.NET demonstrates this data persistence using view state:

Visual Basic **LearningASP\VB\ViewState.aspx**

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  Sub Click(ByVal s As Object, ByVal e As EventArgs)
    messageLabel.Text = nameTextBox.Text
  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>View State Example</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:TextBox ID="nameTextBox" runat="server" />
    <asp:Button ID="submitButton" runat="server"
```

```
        Text="Click Me" OnClick="Click" />
    <asp:Label ID="messageLabel" runat="server" />
  </div>
  </form>
</body>
</html>
```

C# **LearningASP\CS\ViewState.aspx**

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  void Click(Object s, EventArgs e)
  {
    messageLabel.Text = nameTextBox.Text;
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>View State Example</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:TextBox id="nameTextBox" runat="server" />
      <asp:Button id="submitButton" runat="server"
          Text="Click Me" OnClick="Click" />
      <asp:Label id="messageLabel" runat="server" />
    </div>
  </form>
</body>
</html>
```

In this case, the code uses ASP.NET controls with the `runat="server"` attribute. As you can see in Figure 2.3, the text from the box appears on the page when the button is clicked, but also notice that the data remains in the text box! The data in this example is preserved by view state.

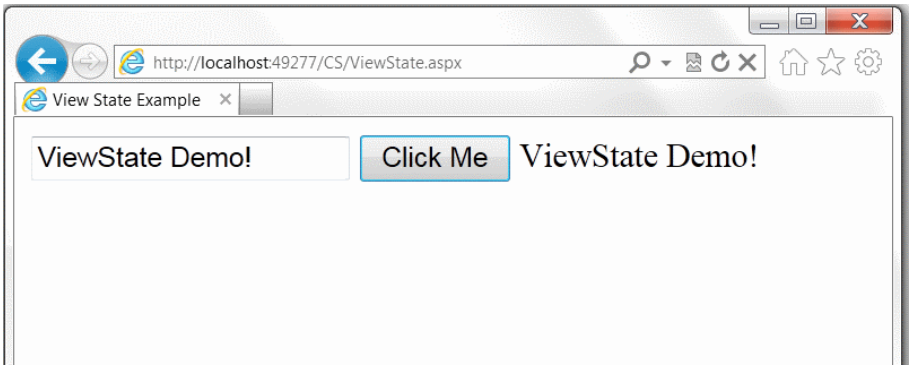

Figure 2.3. ASP.NET maintaining the state of the controls

You can see the benefits of view state already. But where's all that information stored?

ASP.NET pages maintain view state by encrypting the data within a hidden form field. View the source of the page after you've submitted the form, and look for the following code:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
    value="/wEPDwUKLTEwNDY1NzgOMQ9…OfMCR+FN5P6v5pkTQwNEl5xhBk" />
```

This is a standard HTML hidden form field. All information that's relevant to the view state of the page is stored within this hidden form field as an encrypted string.

View state is enabled for every page by default.Using view state comes with a slight performance penalty due to the additional HTML being emitted by the server. If you don't intend to use view state, it is recommended that you disable it. To do this, set the `EnableViewState` property of the `Page` directive to `false`:

```
<%@ Page EnableViewState="False" %>
```

Disabling View State, Control by Control

View state can also be disabled for particular controls in a page: simply set their `EnableViewState` property to `false`. We'll see working examples of this in the following chapters.

Another new feature in ASP.NET 4 is the `ViewStateMode` feature. This allows you to turn off view state for an entire page, but allows for a control to override it. `ViewStateMode` is enabled in the page directive, and it only works if `EnableViewState` is set to true. If `EnableViewState` is set to false, all view state is disabled, regardless of the setting of the `ViewStateMode`. `ViewStateMode` has three settings—Inherit, Enabled, or Disabled. Consider the following code:

```
<%@ Page EnableViewState="True" ViewStateMode=”Disabled” %>
<asp:Label id=”messageLabel” runat=”server” ViewStateMode=”Enabled”
  />

<asp:Label id=”errorLabel” runat=”server” ViewStateMode=”Inherit” />
<asp:Label id=”copyrightLabel” runat=”server” ViewStateMode=➥
  ”Disabled” />
```

Because view state is enabled through the page directive (`EnableViewState=”true”`), the ViewStateMode directive is allowed to be used. In our example, it is set to Disabled. Therefore, all controls on the page will have their view states disabled unless it is specifically overridden. The `messageLabel` control does just so, and it will have view state. The next control, `errorLabel`, is chosen to inherit the settings of the parent, which is the page's directive of Disabled: therefore, it will not have view state enabled. Inherit is also the default option of all controls if an option isn't specifically chosen. The final `copyrightLabel` control has its view state specifically disabled. This allows finer control over your view state. You can now disable view state for your entire page except for one control without having to change every single control on your page. Speaking of page directives, it's time to take a closer look at these curious beasts!

Working with Directives

For the most part, ASP.NET pages resemble traditional HTML pages with a few additions. In essence, just using the **.aspx** extension for an HTML file will ensure that IIS passes the page to the .NET Framework for processing. However, before you can work with certain, more advanced features, you'll need to know how to use directives.

We talked a little about directives and what they can do earlier in this chapter. You learned that directives control how a page is created and cached, help with bug-

fixing, and allow us to import advanced functionality for use within our code. Three of the most commonly used directives are:

`Page`
: This directive defines page-specific attributes for the ASP.NET page, such as the language used for server-side code. We've already seen this `Page` in use.

`Import`
: The `Import` directive makes functionality that's been defined elsewhere available in a given page. The following example, for instance, imports functionality from the `System.Web.Mail` namespace, which you could use to send email from a page. Namespaces are simply .NET's way of keeping all its functionality neatly organized (we'll see how they work in Chapter 3):

```
<%@ Import Namespace="System.Web.Mail" %>
```

You'll become very familiar with this directive as you work through this book.

`Register`
: This directive allows you to register a user control for use on your page. We'll cover `Register` in Chapter 4, but the directive looks something like this:

```
<%@ Register TagPrefix="uc" TagName="footer"
    Src="footer.ascx" %>
```

ASP.NET Languages

As we saw in the previous chapter, .NET supports many different languages. If you're used to writing ASP 2.0 or ASP 3.0, you may think the choice of VBScript or JScript would be an obvious one. But, with ASP.NET, Microsoft did away with VBScript, merging it with Visual Basic. ASP.NET's support for C# is likely to find favor with developers from other backgrounds. By the end of this section, you'll likely agree that the similarities between the two are astonishing—any differences are minor and, in most cases, easy to figure out.

Traditional server technologies are much more constrained in terms of the development languages they offer. For instance, old-style CGI scripts were typically written with Perl or C/C++, JSP uses Java, Coldfusion uses CFML, and PHP is a technology and a language rolled into one. .NET's support for many different languages lets

developers choose the ones they prefer. To keep things simple, this book will consider the two most popular: VB and C#. You can choose the language that feels more comfortable to you, or stick with your current favorite if you have one.

Visual Basic

The latest version of Visual Basic is the result of a dramatic overhaul of Microsoft's hugely popular Visual Basic language. With the inception of Rapid Application Development (RAD) in the 1990s, Visual Basic became extremely popular, allowing in-house teams and software development shops to bang out applications hand over fist. The latest version of VB has many advantages over older versions, most notably the fact that it has now became a fully object oriented language. At last, it can call itself a true programming language that's on a par with the likes of Java and C++. Despite the changes, VB generally stays close to the structured, legible syntax that has always made it so easy to read, use, and maintain.

C#

The official line is that Microsoft created C# in an attempt to produce a programming language that coupled the simplicity of Visual Basic with the power and flexibility of C++. However, there's little doubt that its development was at least hurried along by Microsoft's legal disputes with Sun. After Microsoft's treatment (some would say abuse) of Sun's Java programming language, Microsoft was forced to stop developing its own version of Java, and instead develop C# and another now defunct language, J#. C# is currently one of the most popular in-demand languages by employers because the syntax is very much like what is taught in universities, as well as being easy to pick up by C++ developers.

Summary

In this chapter, we started out by introducing key aspects of an ASP.NET page including directives, code declaration blocks, code render blocks, includes, comments, and controls. We took a closer look at the two most popular languages that ASP.NET supports, which we'll be using throughout this book. In the next chapter, we'll create a few more ASP.NET pages to demonstrate form processing techniques and programming basics, before we turn our attention to the topic of object oriented programming for the Web.

Chapter 3

VB and C# Programming Basics

One of the great things about using ASP.NET is that we can pick and choose which of the various .NET languages we like. In this chapter, we'll look at the key programming principles that will underpin our use of Visual Basic and C#. We'll start by discussing some of the fundamental concepts of programming ASP.NET web applications using these two languages. We'll explore programming fundamentals such as variables, arrays, functions, operators, conditionals, loops, and events, and work through a quick introduction to object oriented programming (OOP). Next, we'll dive into namespaces and address the topic of classes—seeing how they're exposed through namespaces, and which ones you'll use most often.

The final sections of the chapter cover some of the ideas underlying modern, effective ASP.NET design, including code-behind and the value it provides by helping us separate code from presentation. We finish with an examination of how object oriented programming techniques impact upon the ASP.NET developer.

Programming Basics

One of the building blocks of an ASP.NET page is the application logic: the actual programming code that allows the page to function. To get anywhere with ASP.NET,

you need to grasp the concept of **events**. Most ASP.NET pages will contain controls such as text boxes, checkboxes, and lists. Each of these controls allow the user to interact with the application in some way: checking checkboxes, scrolling through lists, selecting list items, and so on. Whenever one of these actions is performed, the control will raise an event. It's by handling these events within our code that we get ASP.NET pages to do what we want.

For example, imagine that a user clicks a button on an ASP.NET page. That button (or, more specifically, the ASP.NET `Button` control) raises an event (in this case, it will be the `Click` event). A method called an **event handler** executes automatically when an event is raised—in this case, the event handler code performs a specific action for that button. For instance, the `Click` event handler could save form data to a file, or retrieve requested information from a database. Events really are the key to ASP.NET programming, which is why we'll start this chapter by taking a closer look at them.

It wouldn't be practical, or even necessary, to cover *all* aspects of VB and C# in this book, so we're going to discuss enough to get you started, and then complete this chapter's projects and samples using both languages. Moreover, the programming concepts you'll learn here will be more than adequate to complete the great majority of day-to-day web development tasks using ASP.NET.

Control Events and Subroutines

As I just mentioned, an event (sometimes more than one) is raised, and handler code is called, in response to a specific action on a particular control. For instance, the code below creates a server-side button and label. Note the use of the `OnClick` attribute on the `Button` control. If you want to test the code, save the file in the **LearningASP** directory you've been using for the other examples. Here's the VB version:

Visual Basic — LearningASP\VB\ClickEvent.aspx

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
  Public Sub button_Click(ByVal s As Object, ByVal e As EventArgs)
```

```
    messageLabel.Text = "Hello World"
  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>Click the Button</title>
  </head>
  <body>
    <form id="form1" runat="server">
    <div>
      <asp:Button ID="button" runat="server"
          OnClick="button_Click" Text="Click Me" />
      <asp:Label ID="messageLabel" runat="server" />
    </div>
    </form>
  </body>
</html>
```

Here's an excerpt from the C# version:

C# LearningASP\CS\ClickEvent.aspx *(excerpt)*

```
<%@ Page Language="C#" %>
⋮
<script runat="server">
  public void button_Click(Object s, EventArgs e)
  {
    messageLabel.Text = "Hello World";
  }
</script>
⋮
```

The HTML and ASP.NET elements in the page are the same for both the C# and VB versions of the code, so I've shown only the differences in the code listing for the C# version above. This approach will be used for the other examples in this chapter, too. The complete C# version can be found in the code archive.

When the button's clicked, it raises the `Click` event, and ASP.NET checks the button's `OnClick` attribute to find the name of the handler subroutine for that event. In the previous code, we instruct ASP.NET to call the `button_Click` routine. You'll see the code for the `button_Click` routine within the `<script>` code declaration block:

Visual Basic LearningASP\VB\ClickEvent.aspx *(excerpt)*

```
<script runat="server">
  Public Sub button_Click(ByVal s As Object, ByVal e As EventArgs)
    messageLabel.Text = "Hello World"
  End Sub
</script>
```

C# LearningASP\CS\ClickEvent.aspx *(excerpt)*

```
<script runat="server">
  public void button_Click(Object s, EventArgs e)
  {
    messageLabel.Text = "Hello World";
  }
</script>
```

This code simply sets a message to display on the `Label` element that we declared with the button. So, when this page is run, and users click the button, they'll see the message "Hello World" appear next to it, as shown in Figure 3.1.

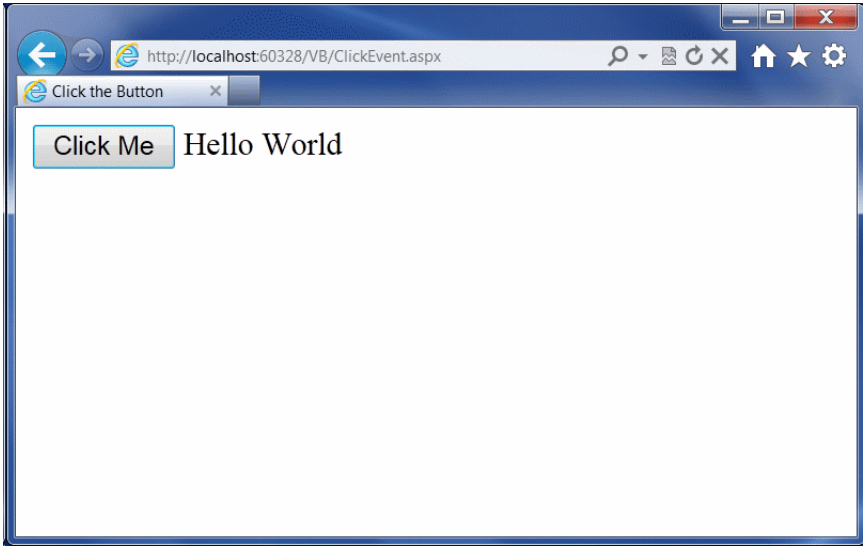

Figure 3.1. Handling the `Click` event

By now, you'll be starting to come to grips with the idea of events, and the ways in which they're used to call particular subroutines. In fact, there are many events that your controls can use, though some of them are found only on certain controls.

Here's the complete set of attributes that the `Button` control supports for handling events:

`OnClick`
: As we've seen, the subroutine indicated by this attribute is called for the `Click` event, which occurs when the user clicks the button.

`OnCommand`
: As with `OnClick`, the subroutine indicated by this attribute is called when the button is clicked.

`OnLoad`
: The subroutine indicated by this attribute is called when the button is loaded for the first time—usually when the page first loads.

`OnInit`
: When the button is initialized, any subroutine given in this attribute will be called.

`OnPreRender`
: We can use this attribute to run code just before the button is rendered.

`OnDisposed`
: The subroutine specified by this attribute is executed when the button is released from memory.

`OnDataBinding`
: This attribute fires when the button is bound to a data source.

Don't worry too much about the details of all these events and when they occur; we just want you to understand that a single control can produce a number of different events. In the case of the `Button` control, you'll almost always be interested in the `Click` event; the others are only useful in rather obscure circumstances.

When a control raises an event, the specified subroutine (if one is specified) is executed. Let's take a look at the structure of a typical subroutine that interacts with a web control: