CHAPTER

4

Introduction to Web Application Development

With great power comes great responsibility.
Spider-Man, Amazing Fantasy Comics #15

In the last three chapters, we've discovered the new features, functions, and promise that Microsoft .NET brings us. Although the .NET platform expands the Rapid Application Model to the Web very well, you need a basic awareness of the technologies being used behind the scenes. As you learned in Chapter 3, overuse of object-oriented programming is not a good thing; neither is a lack of understanding about Web development. We need to learn how to use the powerful new features appropriately so that we can focus on making our applications more efficient and scalable.

Up to this point, we've explored the powerful new features of Microsoft .NET, ASP.NET, and Visual Basic .NET. Now it's time to learn about the fundamentals of building Web applications. These concepts are crucial for efficient and effective development. In this chapter, we will focus on the differences between traditional Visual Basic programming and Web development, the underlying technologies that make the Web work, and the new HTML controls and designer in Visual Studio .NET.

You might be tempted to ask "Why bother learning this when I can just drop a Web control on a Web form and never use this stuff?" The answer is that there will be situations in which you will need an understanding of the architecture and general Web-related technologies to build more powerful applications.

WHY YOU'VE GOT JUST 15 SECONDS...

The Web paradigm adds a whole new level of complexity to traditional application development. The phrase "You've got just 15 seconds to grab a user's attention" is fast becoming a cliché, but it really sums up the idea. You're no longer just creating an application, but a commercial or a TV show, and the user has his or her hand on the remote. It's called surfing the Web for a reason. As a developer, you need to be keenly aware of the impact of architectural and implementation decisions on application performance and scalability. With this technology, as we are discovering, we are still dealing with a client/server model.

The Importance of Protocols

Have you ever seen a diplomatic meeting on a news report? Usually these events are very formal affairs with rules that were clearly defined beforehand. For diplomats, protocol is everything. It specifies the ceremony and etiquette and generally governs every aspect of interaction between nations. In short, it's all about communication. For standards-based communication, protocols are everything as well. Internet protocols are agreed-upon standards for exchanging data between networks on diverse platforms and different environments.

Although understanding the entire network model is important, you'll spend most of your time as a Web developer using the application-level protocols, such as Hypertext Transfer Protocol.

Hypertext Transfer Protocol

HTTP is known as a stateless protocol. It's also one of the most highly used protocols in the process/application layer. Using HTTP for communication is normally divided into two parts: a *request* by the browser (or other client) to a server for information and a *response* by the server fulfilling the client request. This flow should be familiar to you because it resembles the client/server model with which we are all familiar.

HTTP uses Uniform Resource Locators (URLs) to assist in locating documents on Web servers. A URL is associated with a lower-level IP address and can be thought of as a human-readable way to access resources. We'll explore how the two identifiers are resolved later. For now, let's focus on the URL and its successive elements.

The URL Nickel Tour

Normally an HTTP URL or Web address has a form similar to:

```
http://www.gasullivan.com/ourbusiness/ourbusiness.asp
<-1->  <-      2      -> <-          4             ->
```

If you're at or near your computer while reading this, type in this URL and then we'll continue by tearing it apart. Your page should look similar to Figure 4.1.

The first part of the URL is the *protocol*. The second part is the *hostname*, which specifies the domain or server that contains the resources that we want to access. This is also known as a Fully Qualified Domain Name (FQDN). The last part specifies the *document path*, which is a similar concept to locating files on your computer's hard drive. This section is optional as well. Let's look at the next URL example for the final section, also optional, which specifies the *query string*. The query string is used to input parameters for dynamic search or information retrieval.

```
http://quote.yahoo.com/q?s=jwa&d=c
                         <-  5  ->
```

URLs may also be relative, in a similar way to DOS relative paths.

Figure 4.1 The URL (request) and Web page (response).

WHAT IS ICANN?

ICANN is the Internet's technical logistics body. Created in October 1998, ICANN manages the assignment of the following globally unique identifiers in order for the Internet to function:

- **Internet domain names**
- **IP address numbers**
- **Protocol parameter and port numbers**

We talked earlier about client requests and server responses in HTTP. As we begin to investigate this, let's also discover how the URL is resolved with its associated IP address along the way. When you type in a URL that is local to your network and hit Enter, the browser communicates with a server on your network that is running a domain name system service (DNS). DNS is responsible for reconciling the URL with the IP address of the server that you want to access. When you access an external URL, the URL is transmitted to the DNS service managed by interNIC, a group of organizations that have banded together to provide management and guidance for the Internet. (These responsibilities are in the process of transition to the *Internet Corporation for Assigned Names and Numbers, or ICANN.)* It is then resolved with its matching IP address in the organization's DNS registry, and then the request is transmitted to the destination Web server for a response.

The Anatomy of an HTTP Message

An HTTP message begins with a header. It lists the information that is necessary for the transport of the body, which holds the displayable content.

```
When you click Enter to submit the URL, the browser actually sends an
HTTP Request.
```

An HTTP request has two lines. The first line in an HTTP request is called a *request-line*. The request line contains an HTTP user command followed by the HTTP header entry.

HTTP user commands are used to manipulate objects on a server. Each command is associated with a URL that indicates the object's location. Let's examine four of the most relevant HTTP user commands: GET, POST, PUT, and DELETE.

GET. Gets an object from the server. When you want to retrieve a Web page from a server, you GET it. The information is sent as parameters in the query string of the URL.

POST. Used to transfer information from the FORM tag back to the server. Remember, nothing in HTML can be seen by the server unless it's on the form. As we will see, that changes in .NET with the use of the RUNAT=SERVER.

PUT. Allows you to upload information to the server as well. It differs from POST in that the information is sent.

DELETE. The mechanism by which resources on a server are deleted.

Here's the HTTP Request header that results from our URL example:

```
GET /ourbusiness/ourbusiness.asp/ HTTP 1.1
Host: gasullivan.com
```

In this example, GET is the HTTP User command, /ourbusiness/ourbusiness.asp is the document request that the browser is making, and HTTP 1.1 signifies the version of HTTP supported by the browser.

The second line in an HTTP request is an HTTP header entry, which has the following syntax:

```
Name: parameter
```

where name is Host and parameter is gasullivan.com. The host header entry, gasullivan.com, denotes the host domain where the requested document resides.

The HTTP response header provides a lot of information. The following is the HTTP response header to our URL:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://www.gasullivan.com/Default.htm
Date: Sun, 22 Apr 2001 03:10:09 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Wed, 07 Feb 2001 16:18:21 GMT
ETag: "5cd5c1962191c01:973"
Content-Length: 213
```

TIP **If you'd like to quickly see what HTTP response header information is generated for a URL, go to networktools.com. You can type in either a URL or an IP address and get back all manner of interesting information.**

Let's break down the example and investigate the elements. Note that this is not an inclusive list of HTTP headers.

HTTP/1.1 200 OK The first line in a HTTP response message is called the status line. In this example, line 1 specifies that the browser's version of HTTP is supported by the server and that the request succeeded.

The rest of the content in the header is made up of \header messages, many of which are optional.

Server: Microsoft-IIS/5.0 Line 2 specifies the Web service running on the server.

Content-Location: http://www.microsoft.com/Default.htm Line 3 specifies the Fully Qualified Domain Name, document path, and requested document.

Date: Sun, 02 Apr 2001 02:25:37 GMT Line 4 specifies the date of the request.

Content-Type: text/html Line 5 specifies the content type of the document. In this example, it is text-based in an HTML format.

Accept-Ranges: bytes Line 6 notifies the client that the server can accept byte ranges or portions of Web pages.

Last-Modified: Sat, 01 Apr 2001 00:00:37 GMT Line 7 specifies the date that the requested document was modified.

ETag: "467d6a18f6c9c01:872" Line 8 is an identifier that is used for specific resources. This allows the server to know the exact document location even if the server is hosting multiple sites within Web farms or the content is based on geographic location. It's simply another way to store needed state information quickly and effectively.

Content-Length: 18556 Line 9 specifies the body content length.

The last line of the header is always a blank line to separate the header from the body.

As you can see from the previous example, quite a bit of information can be passed between the client and server during the request and response. Fortunately for you, you're isolated from this lower level when using Visual Basic .NET and ASP.NET, but you can still access the information itself through the ASP.NET object model. We look at this in more detail later in the book. If you want to read more or study further about HTTP itself, check out *HTTP Essentials* by Stephen Thomas (Wiley Computer Publishing, 2001).

If you are more interested in a discussion of how lower-level Internet programming (for example, working directly with TCP/IP through sockets) relates to Visual Basic .NET, take a look at *Visual Basic .NET Internet Programming* by Carl Franklin (Wiley Computer Publishing, 2002).

One last thought before we move on. As you've probably guessed by now, there isn't a mechanism for maintaining state built into HTTP. We'll explore later some techniques that you can use to compensate for this.

Now that we have examined the header information in both the request and the response, we need to examine the body. And to do that, it's a good idea to understand how, as the body of the message is sent in HTML, dynamic HTML works and explore some examples of its use.

Dynamic HTML Support within ASP.NET (HTML 4.0)

As discussed in the introduction, I assume you have some experience with HTML. If not, check out the links page at the Web site for this book at wiley.com for some great HTML resources, or if you just like the feel of a good book, pick up a copy of *The Project COOL Guide to HTML* by Teresa A. Martin (John Wiley & Sons, 1996). That said, you might be wondering how a client-side technology such as dynamic HTML and a server-side technology such ASP.NET can interact in a way that will benefit you as developer.

HTML is very good at what it does, namely, marking up a document so that the elements will display correctly. That's all well and good, unless you want your Web site to respond to requests or user input on the fly. Every change means a round trip to the server to process and display the updated results. What a Web developer needs is a way to process some of these changes within the browser itself, thereby conserving resources and improving performance.

Dynamic HTML affords just that through the addition of the Document Object Model to HTML. Now every element in the document is an object (including the document) with properties, methods, and events. For a Visual Basic developer experienced in using objects, this should be old hat. Unfortunately, even though this is now a standard, DHTML is still implemented differently in Internet Explorer and Netscape. For an exhaustive guide to HTML 4.0 and the differences between the browsers, check out the *HTML 4.0 Sourcebook* by Ian S. Graham (John Wiley & Sons, 1998).

For our purposes, we'll focus on the extended support and resources provided by ASP.NET to client side technologies. Using it, you have programmatic access to every element in the HTML document. First, we'll explain some of the more common elements, properties, methods, and events. Then it's off to several examples to see how this all ties together.

Now it's time to take what we've learned so far in Chapter 4 and discover how it applies to .NET.

Enter ASP.NET Server Controls

You may be wondering how Web architecture and information flow fits into the overall topic of Web applications and ASP.NET. An understanding of HTML and DHTML is essential before using one of the powerful features of ASP.NET: server controls.

ASP.NET server controls are server components that render as HTML. The intrinsic HTML controls have a one-to-one mapping equivalent in the ASP.NET server controls. Notice that for each of the HTML tags in Table 4.1, there is a corresponding HTML server control. Also, the ASP.NET HTML server controls can use DHTML to accomplish their tasks. Because these are

truly server side, no ActiveX, Java, or client-side script is needed to populate values from the client. As we'll see, they are simply handled in the post back. The key to the HTML server controls is that they run on the server and render themselves as HTML, as appropriate to the client. We can see the expanded list of ASP.NET server HTML controls and their associated tags in Table 4.1.

Keep in mind, that, as you think about using these, the ASP.NET HTML server controls are basically for migration and only need to be used when server-side processing, resources, or programmatic manipulation are required. Otherwise we can use the ASP.NET server controls, which are introduced in Chapter 5 and then are discussed further in Chapter 6.

Table 4.1 HTML Controls versus HTML Tags

SERVER CONTROL	ASSOCIATED HTML TAG
HtmlAnchor	<a>
HtmlButton	<button>
HtmlForm	<form>
HtmlGenericControl	Any unassociated tag, such as , , or <p>
HtmlImage	<img>
HtmlInputButton (Button)	<input type="button">
HtmlInputButton (Reset)	<input type="reset">
HtmlInputButton (Submit)	<input type="submit">
HtmlInputCheckBox	<input type="check">
HtmlInputFile	<input type="file">
HtmlInputHidden	<input type="hidden">
HtmlInputImage	<input type="image">
HtmlInputRadioButton	<input type="radio">
HtmlInputText (Password)	<input type="password">
HtmlInputText (Text)	<input type="text">
HtmlSelect	<select>
HtmlTable	<table>
HtmlTableCell	<td>
HtmlTableRow	<tr>
HtmlTextArea	<textarea>

Accessing the HTML elements through the ASP.NET HTML server controls is very similar to what you've used in Visual Basic.

Event Support Provided by ASP.NET HTML Server Controls

Events are raised and handled on the client by the Event object in traditional Web applications. Because of the separation of the event from its handler, the way events are raised by ASP.NET server controls is different: The events associated with server controls are raised on the client, but the ASP.NET page framework handles them on the server. During server page processing, change events are processed first, without regard for order. When the processing for the change events has occurred, the event that caused the post itself is then processed.

The Event object is crucial to the dynamic part of dynamic HTML. It ties the element to its event handlers. We can also use the Event object to tie specific events (such as those in Table 4.2) on the client to custom event handlers that run on the server.

Of course, you'll still use VBScript and JScript in Visual Studio .NET when you need to do any client-side programming in your Web page, but you'll most likely use Visual Basic .NET to take full advantage of the power of Visual Studio. NET. The control's ID parameter in the page functions just like an object name in Visual Basic. This is what ties the event procedure to the tag. Later in the chapter, we'll see how ASP.NET HTML server controls are used in Visual Basic .NET to help create Rapid Application Development to build Web Applications. Let's take a look at an example:

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="HTMLCONTROLS.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <body MS_POSITIONING="GridLayout">
<h2> NorthWind Employee Page</h2>
    <form id="Form1" method="post" runat="server">
<SELECT id=Select1 name=Select1 runat="server"
onchange="Select1Change()">
<OPTION selected></OPTION>

</SELECT>
<INPUT style="Z-INDEX: 101; LEFT: 105px; POSITION: absolute; TOP: 167px"
type=submit value=Submit id=cmdSubmit onclick="Select1Change()">
     <span id="Demo1">
    </span>
</form>
  </body>
</HTML>
```

Table 4.2 Some Common Events

EVENT	DESCRIPTION
OnClick	The OnClick event is fired when the user clicks the related element.
OnDblClick	The OnDblClick event is fired when the user double-clicks the related element.
OnBlur	The OnBlur event is fired when the object loses focus (similar to the Visual Basic LostFocus event).
OnFocus	The OnFocus event is fired when the object receives the focus (similar to the Visual Basic GotFocus event).
OnKeyDown	The OnKeyDown event is fired when any key is pressed (similar to the Visual Basic KeyDown event).
OnKeyUp	The OnKeyUp event is fired when any key is released (similar to the Visual Basic KeyUp event).
OnKeyPress	The OnKeyPress event is fired when a key with an ASCII equivalent is pressed (similar to the Visual Basic KeyPress event).
OnHelp	The OnHelp event is fired when the user clicks the related element.
OnMouseMove	The OnMouseMove event is fired when any key is pressed (similar to the Visual Basic KeyDown event).
OnMouseOver	The OnMouseOver event is fired when any key is pressed (similar to the Visual Basic KeyDown event).
OnMouseDown	The OnMouseDown event is fired when any key is pressed (similar to the Visual Basic KeyDown event).
OnMouseUp	The OnMouseUp event is fired when any key is pressed (similar to the Visual Basic KeyDown event).
OnLoad	The OnLoad event is fired immediately after the browser loads the object (similar to the Visual Basic Form_Load event).
OnUnload	The OnUnload event is fired immediately before the browser loads the object (similar to the Visual Basic Form_Unload event).
OnReadyStateChange	The OnReadyStateChange event is fired when the object's state has changed.

The following Visual Basic .NET file is associated in the preceding page by the CodeBehind directive at the top of the Web page. It's what ties the two together and enables the server control events to be defined in the Visual Basic .NET file.

In the second part of our example, we are loading the list box from within the Page_Load event. By checking the IsPostBack property of the page, we are assured of only loading the list box when the form is posted. We normally use this to initialize, much like the Form_Load event in Visual Basic 6.

The second thing that this example shows is the Select1Change Event. This server-side event fires when the list box selection is changed on the client. The two in this case are tied together through the OnClick = attribute of the tag in the page example. Although the preceding HTML sample shows the one-to-one relationship between ASP.NET HTML server controls and their corresponding tags, this one shows the functionality that the controls provide:

```
    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
        Dim addList As ArrayList = New ArrayList()
        If Not IsPostBack Then
            addList.Add("Nancy Davolio")
            addList.Add("Andrew Fuller")
            addList.Add("Janet Leverling")
            addList.Add("Linda Peacock")
            addList.Add("Steven Buchanan")
            addList.Add("Linda Peacock")
            addList.Add("Linda Peacock")

            Select1.DataSource = addList
            Select1.DataBind()
        End If
    End Sub
    Public Sub Select1Change()

        Demo1.InnerText = "You chose: " & Select1.Value
    End Sub
```

Event Bubbling

Remember event bubbling in DHTML? Well, now you really don't need to use it nearly as much as in the past because it's used in a more limited way. Event bubbling is used by nested controls (such as buttons within template columns in a data grid row), as opposed to each one raising an event itself. This raises a single generic event called ItemCommand that passes parameters (CommandArgument) that indicate the control that raised the original event. You can set the CommandArgument property to unique values and then use the event handler to capture and act on them accordingly in a single event and avoid having to write an event handler for each control.

Creating a Web Project Using the HTML Designer

For the remainder of the chapter, let's explore the HTML designer and build a user interface. We'll also jump to the associated Visual Basic file and add a bit of code to show the power and one-to-one mapping of the HTML server controls.

Our first step is to create a new Visual Basic project within Visual Studio .NET. This project will familiarize you with the HTML designer, the code-behind files, and how the HTML server tags relate to Visual Basic .NET. First, we want to set it up as follows:

1. Create a Visual Basic ASP.NET Web application on server http://localhost. Name the application HTMLApp1.
2. Right-click on HTMLApp1 in the Solution Explorer window.

The Web form is the object that reflects the HTML document in the browser. Let's think of it, conceptually, as a Visual Basic form. The next step is to set and verify some properties.

Now that we have the Web form page open, we'll want to specify the PageLayout property to make us feel even more at home. We'll also change the targetSchema property to handle older browsers:

1. Select DOCUMENT item in the Properties window drop-down list.
2. Set the PageLayout property to GridLayout. This allows HTML elements to be positioned on the document in a similar fashion to a Visual Basic form.
3. Verify that the targetSchema property is set to Internet Explorer 3.02/Navigator 3.0.

A bland Web page is pretty boring. Let's add some buttons on the page that will show how easy formatting is. Then we'll start to add functionality.

1. Drag two Button elements and the Text Field element from the HTML tab of the toolbox to the HTML page. These are the HTML server controls. Arrange the buttons in a column, with the Text Field next to the first button.
2. Left-click on the topmost button to select it.
3. Hold down the Shift key, and left-click on the other button to it.
4. From the Format menu select Align, then Lefts from the popup menu
5. Press the Escape key on the keyboard to deselect all buttons.

Now that we have added our buttons, let's begin to change some properties. This should seem like old hat. We're going to demonstrate that Web development model is consistent with what we've used before.

1. Select the topmost button. Verify that Button1<INPUT> appears in the drop-down list of the Properties window.
2. In the Properties window set the value attribute to Script and press Enter. Note that the caption on the button changed from Button to Script.
3. Set the ID property to btnScript.
4. Select the bottom button. Verify that Button2 <INPUT> appears in the Properties window drop-down list.
5. In the Properties window set the value attribute to Code and press Enter. Note that the caption on the button changed from Button to Code.
6. Set the ID property to btnCode.

Let's pause for a moment and investigate the HTML source. If we want to ensure that our application can be used in the widest variety of browsers, we need to use HTML 3.2. The table that we'll examine is used to position the buttons for older browsers supporting the HTML 3.2 standard. This allows for consistent formatting and proper placement of our controls when using HTML 3.2.

1. Click the HTML tab at the bottom of HTMLPage1.htm to switch to HTML Source view.
2. Examine the HTML. Note that it contains an HTML <table>.

If our browsers can support a richer level of functionality, we'll use HTML 4.0 instead. Let's make the change now:

1. Click the Design tab at the bottom of HTMLPage1.htm to switch to HTML Design view.
2. In the Properties window drop-down list, make sure that DOCUMENT is selected. In the Properties window, set the targetSchema to HTML 4.0.

We're still being consistent with the Visual Basic Rapid Application Model so that we can place our code in the associated Visual Basic .NET file even though we are programming for the Web. Just as we did in the past, we've got a form with Visual Basic behind it. Let's continue by converting our HTML server controls to run on the server. Then we'll add some code to change some document properties when the button is clicked. The browser still gets the HTML, but we are manipulating it now with Visual Basic .NET. For the first button, we'll change the code in a script tag and use a span tag to show our change, and in the second one, we'll do it in the code window.

1. Left-click on the topmost button to select it.
2. Hold down the Shift key, and left-click on the other button to select it.
3. Right-click and select Run as server control.
4. Press the Escape key on the keyboard to deselect all the buttons.
5. Left-click on the text field to select it.
6. Right-click and select Run as server control.
7. Click the HTML tab at the bottom of the Web form to switch to HTML view.
8. Locate the </HEAD> tag.
9. Just before it add the following code:

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

Sub btnScript_Click(source As Object, e As EventArgs)

        Message.InnerHtml = "Hi From Script Block"



End Sub
```

10. Locate the INPUT id=btnScript tag.
11. Scroll to just before runat=server and type the following:

```
onserverclick="btnScript_Click"
```

12. Locate the FORM tag.
13. Just after it, add the following code:

```
<H1><span ID="Message" runat="server"> </span></HI>
   </script>
```

Now, let's do the same for the second button. Only this time we'll change the property from the code-behind file.

1. Click the Design tab to return to form view.
2. Double-click btnCode to access the code window and the button control's click event handler.
3. In the ServerClick event procedure, add the following code:

```
txtHello.Value = "Hi from CodeBehind"
```

One of the added benefits of HTML 4.0 is dynamic positioning of controls and text within a page. ASP.NET supports this directly. We can use a panel to see how this works:

1. Double-click on the Panel (Linear Layout) control in the HTML tab of the toolbox. A new panel will appear selected in the middle of the document.
2. Drag a Text Field element from the toolbox and drop it in the center of the panel. The Text Field will appear at the top of the panel because of the panel's flow layout rule.
3. Drag and drop two more Text Field elements into the Panel element.
4. Click on an empty part of the panel and drag it to the left side of the HTML page.
5. Resize the panel to the right by dragging the center selection box on the right side of the panel to the size you want. Note that the Text Fields will rearrange (flow) themselves as needed to fill the panel from left to right, top to bottom.

The document outline enables us to keep track of all the elements contained within the document, allowing for efficient movement within. Let's examine the document structure and see how all the elements within our page are laid out in a hierarchical format. We drop the controls on our Web form, which is consistent with how we've programmed in the past, but they're implemented on the client side as HTML.

1. Show the Document Outline by pressing Ctrl-Alt-T.
2. Use the Document Outline to explore the structure of the HTML document (when you click on an item in the Document Outline panel, the corresponding item is highlighted in the Designer).
3. Notice how the panel is implemented using an HTML object and that the three text items are logically contained within the .

Finally, let's see the page we created. We'll save and then view our page in the browser and test the resulting functionality.

1. Save the HTML page using the File menu.
2. Right-click on the HTML and choose View in Browser. Note that the Preview is integrated into Visual Basic. You also have the option to preview in one or more external browsers.
3. Click the buttons and observe that the code is functional.

We've explored basic HTML functionality within Visual Basic .NET. Now for a quick review and then on to Chapter 5.

Wrapping Up

In this chapter, you have learned about Web development fundamentals. We investigated Web architecture and the underlying technologies within the

network communication model. Next, we focused on the HTML server controls that ASP.NET provides and saw how each tag can now be manipulated as an object, complete with server-side event handling.

We rounded out this chapter with a focus on how DHTML fits within both Visual Basic .NET and ASP.NET and worked through some exercises that illustrated how closely the Visual Basic Rapid Application Development model has been ported to Web development. We've learned about the fundamentals of Web Application Development specifically as it relates to ASP.NET and its basic technologies. Now it's time to move into the parts and pieces of ASP.NET, starting with the Page Framework in Chapter 5.