Mastering™ ASP.NET with VB.NET

A. Russell Jones

Chapter 4: Introduction to ASP.NET

ISBN: 0-7821-2875-0

Sybex Inc.
1151 Marina Village Parkway
Alameda, CA 94501
U.S.A.
Phone: 510-523-8233
www.sybex.com

Chapter 4

Introduction to ASP.NET

ASP.NET IS THE .NET framework layer that handles Web requests for specific types of files, namely those with `.aspx` and `.acsx` extensions. The ASP.NET engine provides a robust object model for creating dynamic content and is loosely integrated into the .NET framework. This integration makes it easy to change the implementation when the .NET framework migrates to platforms other than Windows.

In this chapter:

- What is ASP.NET?
- Why do you need ASP.NET?
- What does ASP.NET do?
- Why is ASP.NET in a VB.NET book?
- Creating your first Web Form

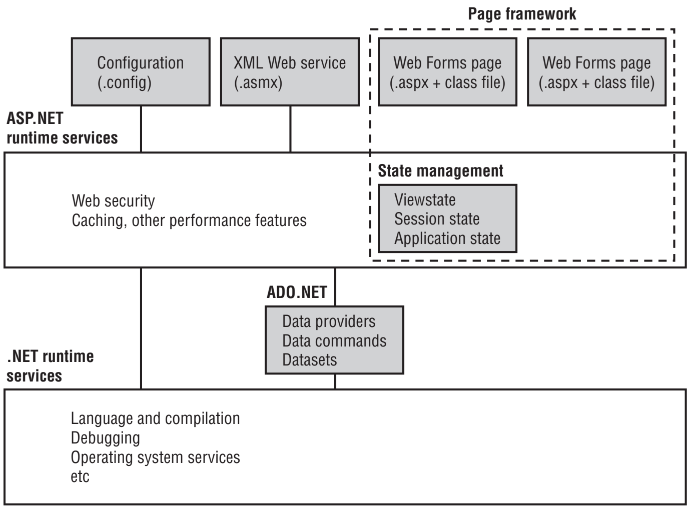
What Is ASP.NET?

What is ASP.NET? This may seem like a relatively simple question, but I assure you that it's not. Because ASP.NET is part of the .NET framework, it is available on any server with the framework installed. In other words, it's not an add-on anymore; ASP has become legitimate. ASP.NET is implemented in an assembly that exposes classes and objects that perform predetermined specific tasks. If you are familiar with "classic" ASP (the versions of ASP that preceded .NET), you'll find that your approach to programming in ASP.NET is somewhat different, but the concepts behind building a Web application are much the same. If you're not familiar with classic ASP, so much the better—you won't have as much information to forget!

ASP.NET programs are centralized applications hosted on one or more Web servers that respond dynamically to client requests. The responses are dynamic because ASP.NET intercepts requests for pages with a specific extension (`.aspx` or `.ascx`) and hands off the responsibility for answering those requests to just-in-time (JIT) compiled code files that can build a response "on-the-fly." Figure 4.1 shows how ASP.NET integrates with the rest of the .NET framework.

FIGURE 4.1

ASP.NET integration with the .NET framework



From looking at Figure 4.1, you can see that ASP.NET deals specifically with configuration (`web.config` and `machine.config`) files, Web Services (ASMX) files, and Web Forms (ASPX) files. The server doesn't "serve" any of these file types—it returns the appropriate content type to the client. The configuration file types contain initialization and settings for a specific application or portion of an application. Another configuration file, called `machine.web`, contains machine-level initialization and settings. The server ignores requests for WEB files, because serving them might constitute a security breach.

This book concentrates on Web Forms and Web Services. Client requests for these file types cause the server to load, parse, and execute code to return a dynamic response. For Web Forms, the response usually consists of HTML or WML. For Web Services, the server typically creates a Simple Object Access Protocol (SOAP) response. While SOAP requests are inherently stateless and can thus execute immediately, Web Forms are stateful by default. Web Forms maintain state by round-tripping user interface and other persistent values between the client and the server automatically for each request. In Figure 4.1, the dashed rectangle titled Page Framework shows the difference—a request for a Web Form can use ViewState, Session State, or Application State to maintain values between requests. It is possible (but not the default) to take advantage of ASP.NET's state maintenance architecture from a Web Service, but for performance reasons, you should generally avoid doing so.

Both Web Forms and Web Services requests can take advantage of ASP.NET's integrated security and data access through ADO.NET, and can run code that uses system services to construct the response.

So the major difference between a static request and a dynamic request is that a typical Web request references a static file. The server reads the file and responds with the contents of the requested file. With ASP.NET there's no such limitation. You don't have to respond with an existing file—you can respond to a request with anything you like, including dynamically created HTML, XML, graphics, raw text, or binary data—anything. Capability, by itself, is nothing new—you've been able to create

CGI programs, JavaServer pages, classic ASP pages, ColdFusion, and NetObjects Fusion pages for quite some time. All these technologies give you the ability to respond to an HTTP request dynamically. So, what are the differences?

- Unlike classic ASP, ASP.NET uses .NET languages. Therefore, you have access to the full power of any .NET assembly or class in exactly the same way as you do from VB.NET. In this sense, ASP.NET is similar to early compiled CGI programs, but with CGI, a separate copy of the program had to be loaded and executed for each request. ASP.NET code exists in multithreaded JIT compiled DLL assemblies, which can be loaded on demand. Once loaded, the ASP.NET DLLs can service multiple requests from a single in-memory copy.
- ASP.NET supports all the .NET languages (currently C#, C++, VB.NET, and JScript, but there are well over 20 different languages in development for .NET), so you will eventually be able to write Web applications in your choice of almost any modern programming language. JavaServer pages support only Java, but because Java now has a wide support base, that's not much of a limitation. Classic ASP supports several scripting language versions (although in practice, VBScript and JScript are by far the most prevalent). The scripting languages let you extend ASP's basic functionality by writing DLLs in any COM-compliant language. ColdFusion uses ColdFusion Markup Language (CFML) tags, which have a powerful but limited set of capabilities; however, you can extend CFML with custom programming.
- Microsoft was able to draw on millions of hours of developer experience with classic ASP, so in addition to huge increases in speed and power, ASP.NET provides substantial development improvements, like seamless server-to-client debugging, automatic validation of form data, and a programming model very similar to that of a Windows application.

Framework for Processing HTTP Requests

Microsoft's Web server, Internet Information Server (IIS), handles HTTP requests by handing the request off to the appropriate module based on the type of file requested. Note that the IIS responds with one of only a few possible actions when it receives a request:

Respond with the file's contents The server locates and reads the requested file's contents and then streams the contents back to the requester. The server responds in this manner to `.htm` and `.html` file requests, as well as to all requests that have no associated application type—for example, EXE files.

Respond by handing off the request The server hands off requests for files that end in `.asp` to the classic ASP processor, and files that end in `.aspx`, `.ascx`, or `.asmx` to the ASP.NET processor.

Respond with an error IIS responds with a customizable error message when a requested file does not exist or when an error occurs during processing.

Classic ASP versus ASP.NET

In classic ASP, the server handed off file requests that ended in `.asp` to the ASP engine, an Internet Server Application Programming Interface (ISAPI) ASP DLL. Because there's a difference in the file extension (`.asp` versus `.aspx`, `.ascx`, and `.asmx`) for classic ASP and ASP.NET files, respectively, you can have both running on the same server simultaneously. Fortunately for ASP programmers, ASP.NET supports all the functionality available in classic ASP and a great deal more besides. Table 4.1 shows the major differences between the two technologies.

Table 4.1: Comparison of Classic ASP and ASP.NET

Classic ASP	**ASP.NET**	**Description**
Intercept client requests for files with an `.asp` extension.	Intercept client requests for files with the `.aspx` extension.	Provides the ability to create content "on-the-fly"—dynamic content.
Write server-side script in one of a small number of languages. Script languages are interpreted at runtime.	Write server-side code in any .NET language. .NET languages are compiled, not interpreted.	Compiled code is faster. The development environments and debug facilities are more powerful.
Extend ASP scripting functionality with COM objects.	Use any of the .NET System classes or call existing COM objects.	Provides the ability to extend ASP capabilities by writing custom code.
All processing happens *after* the server passes control to the ASP engine. Cannot take advantage of ISAPI services.	You can write code to intercept requests *before* the ASP engine takes control. You can write ISAPI services within the .NET framework.	Sometimes, you want to respond to a request *before* the ASP engine parses the request. You can do that in .NET, but not with classic ASP.
Code and HTML are usually mixed in-line within a page.	Code may be placed in-line in ASP.NET pages, but is usually separated from the HTML in "code-behind" files.	The .NET code-behind pages provide a cleaner separation of display and logic code and also simplify code reuse.
Developer responsible for implementing ways to maintain state data between pages.	Web Forms and Web Form controls act much like classic VB forms and controls, with properties and methods for retrieving and setting values.	While both classic ASP and ASP.NET render output in HTML, ASP.NET introduces ViewState, a scheme that automatically maintains the state of controls on a page across round trips to the server. Web Forms, Web Form controls, and ViewState simplify development and eliminate much of the gap between programming Web applications and stand-alone Windows applications.
Process submitted HTML form fields.	Process and validate submitted form fields.	Provides the ability to gather user input. Automatic validation takes much of the grunt work out of programming pages that require user input.

Continued on next page

TABLE 4.1: COMPARISON OF CLASSIC ASP AND ASP.NET *(continued)*

Settings stored in special ASP page that executes code for special events (such as Application startup and shutdown).	Settings stored in XML-formatted files. Settings for subdirectories may override settings for their parent directories.	ASP.NET uses XML files to store settings, giving you programmatic access to configuration settings.
ADO	ADO.NET	ADO.NET is faster, more powerful, and much better integrated with XML for passing data between tiers.
MTS/COM+	Same, through COM interoperability.	VB.NET components can support object pooling, whereas VB6-generated components do not. Eventually, COM+ will be completely integrated into .NET.

Why Do You Need ASP.NET?

The first computer languages were little more than mnemonics substituting for raw machine code instructions, but as computers became more complex, each new language generation has supported an increasing level of abstraction. Visual Basic, for example, abstracted user interface design and construction into simple drag-and-drop operations. For the first time, you could create a working Windows application with very little effort.

Similarly, when Web programming first became widespread, there were few tools to help programmers write Web applications. To create a Web application, you started by writing low-level socket communications code. Over the years, the abstraction level has increased for Web programming as well. ASP.NET is the latest (and arguably the best) of these abstractions, because it lets you work almost exclusively with rich high-level classes and objects rather than directly with raw data. Without ASP.NET, building a Web application is a chore. With ASP.NET, building a Web application is similar to building a Win32 application.

Client Changes

ASP.NET lets you build Web-based applications that interact with pages displayed remotely. Originally, classic ASP was designed to work with browsers, which at that time were capable of little more than displaying data and images wrapped in HTML markup. While the integration hasn't changed, the clients have changed dramatically. For example, modern browsers are much more capable. Not only can they display HTML and images, they also support Dynamic HTML (DHTML), animations, complex image effects, vector graphics, sound, and video—and can run code, letting you offload appropriate portions of your application's processing requirements from your server to the client.

Centralized Web-Based Applications

But it's not only browsers that have changed. Centralized Web-based applications have garnered a huge investment from companies that increasingly need to support mobile and remote clients. The cost of supplying private network connectivity to such clients is prohibitive, yet the business advantages of supporting such clients continue to rise. The only cost-effective way to supply and maintain corporate applications to these mobile and remote workers is to uncouple them from the network and build the applications to work over HTTP through the Internet, WAP, and other advanced protocols. Therefore, Web-based applications are no longer the exclusive purview of Webmasters and specialist developers; they've become an integral part of the corporate IT operations.

Distributed Web-Based Applications

For all the advantages of centralized Web applications, they mostly ignore a huge reservoir of processing power that exists on the client machines. Recently, a new breed of application has begun to attract attention—the *point-to-point* program (often abbreviated as "P-to-P" or "P2P"). These programs typically use XML to pass messages and content directly from one machine to another. Most current implementations, such as Groove and Napster, use a centralized server as a directory service that helps individuals or machines contact one another. Peer-to-peer applications are often called *distributed* because the application runs at many points on the network simultaneously. In addition, the data used by distributed applications is usually (but not necessarily) stored in multiple locations.

Functional Interoperability

As the client transition from stand-alone applications to browser-based interfaces occurred, another factor came into play: interoperability. IT departments have struggled with interoperability ever since programming escaped the confines of the mainframe. As the number of computers and computing devices within the business and entertainment worlds expanded, the problem grew. Today, computing is no longer limited to full-size desktop machines or even laptops. Handheld and notepad computers, telephones, and even pagers communicate with the Web servers and need to display data—sometimes even display the same data or run the same application as a desktop box. Similarly, IT departments now run critical applications on mainframes, minicomputers, and several different types of servers, from small departmental servers to server farms that supply computing power to the entire enterprise and beyond. These servers are made by different vendors and often run differing and incompatible operating systems, yet companies often need to transport and consume data between the various machines, databases, application tiers, and clients.

Companies have attacked the interoperability problem in several ways. They've tried limiting the hardware and software—creating tight corporate standards for desktop, laptop, and handheld computers. That approach hasn't worked very well—the industry changes too fast. They've tried and discarded the thin-client network computer approach. Too little benefit, too late. They've tried implementing Java as both the platform and the language—but performance issues, a lack of cooperation between the major software suppliers, and lack of commercial-quality software have—at least temporarily—quelled that approach as well. Fortunately, a new interoperability standard has recently presented itself—XML.

Standardization, Verifiability, and HTTP Affinity

XML provides a possible solution to some of these interoperability problems. XML is not a panacea, but it does provide a standardized and *verifiable* text-based file format that can help ease the problems involved in moving data from one server to another, as well as accommodate displaying identical data on disparate clients. XML's standardization helps, because the file format is universally recognized. XML simplifies programming because it can verify, by using a *document type definition (DTD)* or *schema,* that a file does indeed contain a specific type of content. Finally, XML's text-based format transfers very well over a plain HTTP connection, which helps avoid problems with firewalls and malicious code.

Web Services

These attributes, standardization, verifiability, and HTTP affinity, led to a new use for ASP—creating server-based code that delivers data without necessarily delivering HTML. In .NET, such pages are called Web Services. You can think of a Web Service as a function call or as an object instantiation and method call across the Web. Just as Web browsers and Web servers use a common protocol, HTTP, to communicate across the network, a Web Service uses a common XML structure, called Simple Object Access Protocol (SOAP) to communicate with the calling application. You'll learn more about SOAP and Web Services in Chapter 21.

What Does ASP.NET Do?

What does ASP.NET do? Again, this is not a simple question. Classic ASP was limited to simple script languages that could respond to requests, but provided no intrinsic direct access to system services other than those few required to read and respond to a request, such as writing output text. While you could extend classic ASP through commercial or custom-built COM components, the relatively high overhead required to create COM objects, and classic ASP's reliance on untyped interpreted scripting languages, limited system performance. In contrast, creating .NET framework objects requires very little overhead, and ASP.NET lets you use fully object-oriented languages with seamless access to system services. Therefore, I'll describe just the primary tasks that ASP.NET accomplishes now, and then fill in the practical details in the remainder of this book.

Accepts Requests

All ASP.NET pages work essentially the same way. A client application makes an HTTP request to a Web server using a URL. The Web server hands off the request to the ASP.NET processor, which parses the URL and all data sent by the client into collections of named values. ASP.NET exposes these values as properties of an object called the HttpRequest object, which is a member of the `System.Net` assembly. An assembly is a collection of classes. Although an assembly *can* be a DLL, it may consist of more than one DLL. Conversely, a single DLL may contain more than one assembly. For now, think of an assembly as a group of related classes.

When a browser, or more properly a *user agent,* makes a request, it sends a string containing type and version information along with the request. You can retrieve the `HTTP_USER_AGENT` string via the HttpRequest object. For example, the following code fragment retrieves several items from the user agent and writes them back to the client. An ASP.NET Web Form Page object exposes the Http-Request with the shorter (and familiar to ASP).

```
Response.Write("UserAgent=" & Request.UserAgent & "<br>")
Response.Write("UserHostAddress=" & Request.UserHostAddress & "<br>")
Response.Write("UserHostName=" & Request.UserHostName & "<br>")
```

Builds Responses

Just as ASP.NET abstracts incoming data in the HttpRequest object, it provides a way to respond to the request via the HttpResponse object. Abstracting responses in this manner has been so successful that you'll find you need to know almost nothing about HTTP itself to use the HttpRequest and HttpResponse objects.

Assists with State Maintenance

Unlike a stand-alone or standard client-server application, Web applications are "stateless," which means that neither the client nor the server "remembers" each other after a complete request/response cycle for a single page completes. Each page requested is a complete and isolated transaction, which works fine for browsing static HTML pages but is the single largest problem in constructing Web applications.

Classic ASP introduced the idea of a *session,* which begins the first time a client requests any page in your application. At that point, the ASP engine created a unique cookie, which the browser then accepted and returned to the server for each subsequent page request. ASP used the cookie value as a pointer into data saved for that particular client in an object called the Session object. Unfortunately, because the client data was stored in memory on a single server, this scheme did not scale well, nor was it fault-tolerant. If the Web server went down, the users lost the in-memory data.

ASP.NET uses much the same cookie scheme to identify specific clients, but the equivalent of the Session object is now called the HttpSessionState object. ASP.NET addresses the session-scalability and data-vulnerability problems in classic ASP by separating state maintenance from the ASP.NET engine. ASP.NET has a second server application, called the Session server, to manage Session data. You can run the Session server in or out of the IIS process on the same machine as your Web server or out of process on a separate computer. Running it on a separate computer lets you maintain a single Session store across multiple Web servers. ASP.NET also adds the option to maintain state in SQL Server, which increases fault tolerance in case the Session server fails.

Why Is ASP.NET in a VB.NET Book?

VB6 had a project type called an IIS Application—a technology more commonly known as Web-Classes. I wrote a book about using WebClasses, called the *Visual Basic Developer's Guide to ASP and IIS* (Sybex, 1999). Using WebClasses, a VB programmer had access to the ASP intrinsic objects—Request, Response, Server, Application, and Session—and could use the compiled code within Web-Classes to respond to client Web requests. But IIS Applications required ASP to be installed on the server and, in fact, were called as COM components from an automatically generated ASP page. Therefore, a WebClass-based application in VB6 was really an ASP application that followed a specific track to instantiate and use VB COM components. Although the entire underlying technology has changed, that aspect has not.

Tip *A VB.NET Web Application project* is *an ASP.NET application!*

ASP.NET, although advertised as if it were a separate technology, is not. It is part of, and completely dependent on, the .NET framework (see Figure 4.1). In fact, an ASP.NET project is *exactly the same thing* as a VB.NET Web Application project. You'll hear that you can write an ASP.NET application using Notepad—and you can! You can also write a VB.NET application using Notepad. But the big advantage of writing a VB.NET application within the Visual Studio .NET (VS.NET) IDE is that you have access to a number of productivity tools, including syntax highlighting, IntelliSense, macros and add-ins, the ToolBox, HTML, XML, code editors, the Server Explorer, etc., etc., etc. Remember that when you create a VB.NET Web Application project, you're really creating an ASP.NET project—you're just approaching the technology through a specific language and IDE.

VB.NET Provides Code-Behind

In an ASP.NET application, you can either write code in-line, as with classic ASP, or you can place the HTML code in a file with an `.aspx` extension and the code in a separate file with an `.aspx.vb` extension, called a *code-behind module* or *code-behind class*. There's little or no difference in performance between the two methods, but there's a fairly large difference in maintenance costs and reusability between the two approaches. For example, Listing 4.1 shows that you can still write code embedded in HTML in a manner very similar to the classic ASP style.

Listing 4.1: Classic ASP embedded code (*ch4-1.aspx*)

```
<%@ Page Language="vb" AutoEventWireup="false"%>
<html>
  <head>
    <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  </head>
  <body>
    <%Response.write("Hello world")%>
    <form id="ch4-1" method="post" runat="server">
    </form>
  </body>
</html>
```

Alternatively, you can create exactly the same output using a code-behind class by placing the line `Response.Write("Hello world")` in the `Load` event for a Web Form (see Listing 4.2). Don't worry if that doesn't exactly make sense at the moment—it will at the end of this chapter.

LISTING 4.2: CODE BEHIND A WEB FORM EXAMPLE (*CH4-2.ASPX.VB*)

```
' VS.NET autogenerated code omitted
Protected Sub Page_Load(ByVal Sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.Load
   If Not IsPostBack Then
      ' Evaluates to true the first time client requests the page
      Response.Write("Hello World")
   End If
End Sub
```

.NET's Unified Platform for Design, Coding, and Debugging

You may already be familiar with the event model for Windows Form controls. One of the goals of .NET was to create that same sense of programmatic unity in working with Web applications. Therefore, even though there's usually a physical separation between the control in a browser and the application on a server, you can often develop pages as if that distance were not present. In .NET, HTML pages containing code are called Web Forms. Don't be misled by the name—both of the preceding code examples are "Web Forms" even though one looks exactly like an HTML page with two lines of code inserted, and the other looks exactly like a VB.NET subroutine.

For those of you who might have used WebClasses (the class type from IIS Web Application projects in VB6), a Web Form (with code-behind) is similar to the combination of an HTML template and a WebClass. The Web Form contains the HTML layout, while the code-behind class contains the program logic and exposes page-level events. But don't be confused—a Web Form is much more powerful than a WebClass.

In the VS.NET IDE, you design a Web Form in much the same way you design a Windows Form—by dragging and dropping controls from the Toolbox onto the Web Form drawing surface. When you add a new Web Form to your project in VB, you can elect to use the drag-and-drop metaphor, or if you're more comfortable editing HTML directly, you can click a tab and move into the HTML text-mode editor.

Because Web Forms aren't Windows Forms, you need to select a target client type. The first VS.NET release lets you target HTML 3.2–compliant clients (Internet Explorer (IE) version 3.*x* and earlier browsers, Netscape version 4.*x* and earlier) or HTML 4 ones (IE 4.*x* and 5.*x*, Netscape 6). Unless you have good reason to support the earlier browsers, you should make sure you target the HTML 4 clients. You can lay out a Web Form in either FlowLayout mode or in GridLayout mode. These two settings control how and where the browser places controls on the page.

When you select the FlowLayout option, the browser uses its standard internal HTML layout rules to place the controls. In other words, it places the controls linearly, from left to right, top to bottom, wrapping to the next line where necessary and exceeding the viewable width and height of the browser where wrapping is not possible, adding scroll bars as needed.

In contrast, GridLayout mode lets you place controls at fixed positions on the page. GridLayout mode is equivalent to writing an HTML `input` control where the `style` attribute specifies the `position:absolute` cascading style sheet (CSS) style. In fact, that's exactly what it does—for example,

```
<input
type="text"
style="position: absolute;
left: 10;
top: 10">
```

After placing controls and content on the Web Form, double-click any control (if necessary, click OK in response to the prompt regarding conversion to a server-based control). VS.NET will open the code-behind module. If you insist on writing embedded code, click the HTML tab at the bottom of the Web Form window and VS.NET will switch to the HTML text editor. You can insert code by enclosing it between `<%` and `%>` tags or between `<script language="VB" runat="server"> </script>` tags.

NOTE *Use the* `<% %>` *syntax only for* in-line *code—code that you want to execute when the page is rendered. Code written in this manner is a* code render block*. To display a variable or the result of an expression, you can use the shorthand* `<%=var or expression%>`*. For all other embedded code, use the* `<script></script>` *syntax. You must declare page-level variables, subroutines, and functions within* `<script></script>` *blocks. You can reference external code using the* `src` *attribute of the* `<script>` *tag. You must include the* `runat="server"` *attribute for all server-side code.*

The ASP.NET engine treats all content on the page that is *not* between those tags as HTML content and streams it directly to the browser. You'll see a few examples of in-line code and code within `<script runat="server">` blocks in this book, but not many, because this book discusses VB.NET-generated code-behind modules almost exclusively.

Creating Your First Web Form

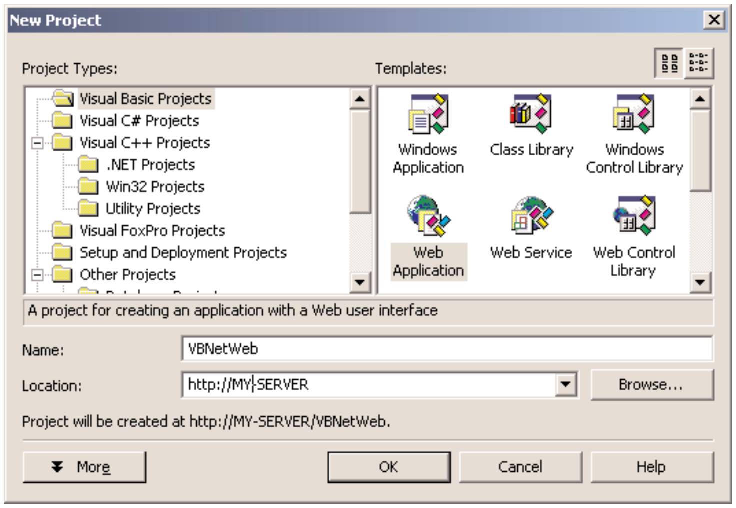
In this section, you'll create a Web Form that lets you enter some text into a textbox—nothing fancy here. You click a Submit button to send the text you enter to the server. But then I'll show you just a tantalizing glimpse of how VB.NET exceeds both VB6 and classic ASP in terms of power. The server will respond with a GIF image, created on-the-fly, containing the text you type into the text box.

Step 1: Creating a Project

Launch VS.NET. Click File ➢ New ➢ Project and select the item Visual Basic Projects in the left pane of the New Project dialog (see Figure 4.2). In the right pane, select the Web Application icon (you may need to scroll to see the icon).

By default, VB.NET names your Web Application projects "WebApplication" with an appended number, for example, WebApplication1, but you should always enter a specific name. Click in the Name field and enter **VBNetWeb**. Check the Location field; it should contain the name of the Web server you want to host this application. Typically, this will read http://localhost. However, you may create a project on any Web server for which you have sufficient permissions to create a virtual directory and write files.

FIGURE 4.2
VS.NET New Project window



A project for creating an application with a Web user interface

Name: VBNetWeb

Location: http://MY-SERVER

Project will be created at http://MY-SERVER/VBNetWeb.

Make sure the information you entered is correct, and then click OK. VS.NET will create the new project.

You should see the Solution Explorer pane (see Figure 4.3). If the Solution Explorer is not visible, select View ➢ Solution Explorer from the menu bar.

You'll use the VBNetWeb project throughout this book. When VB.NET creates a Web Application project, it adds several items to the Solution Explorer pane. I'll explain all these in a minute, but first, create a new folder named `ch4`. Creating subfolders works exactly like creating subfolders in a Web site: you simply add the name of the folder to the root URL to view a page in that folder. To create the subfolder, right-click the VBNetWeb virtual root in the Solution Explorer, click Add, and then click New Folder. Finally, type **`ch4`** for the folder name.

Select the `Web Form1.aspx` file and drop it on top of your new `ch4` folder. When you drop the file, VS.NET will move the ASPX file (and any associated files) into the `ch4` folder. If the file is already open, close it first, and then move it. Your Solution Explorer pane should look similar to Figure 4.4.

Step 2: Laying Out the Page

Select the `Web Form1.aspx` file and then right-click it to bring up the context menu. Select Rename from the menu and rename the file to `DynamicImage.aspx` (don't forget or mistype the extension—it's required).

TIP *You can press F2 to rename a file, just as you can in Windows Explorer.*

Double-click the `DynamicImage.aspx` file to open it in the editing pane. By default, ASPX pages open in Design mode. If you're not in Design mode, click the Design tab at the bottom of the editing window to complete this example.

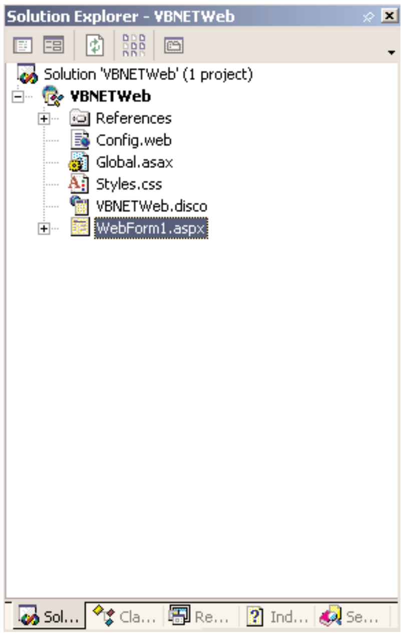


FIGURE 4.3

Solution Explorer pane containing a new project

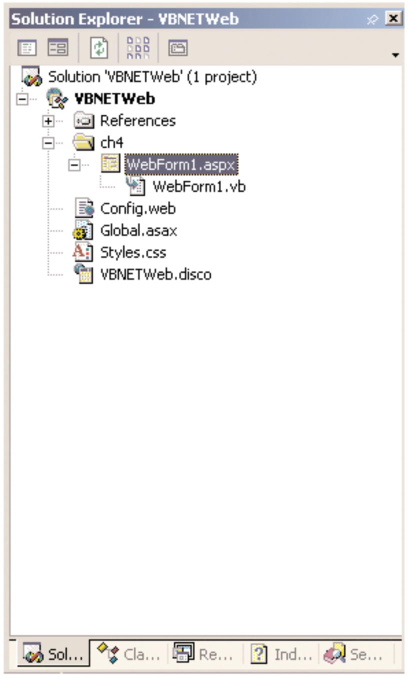


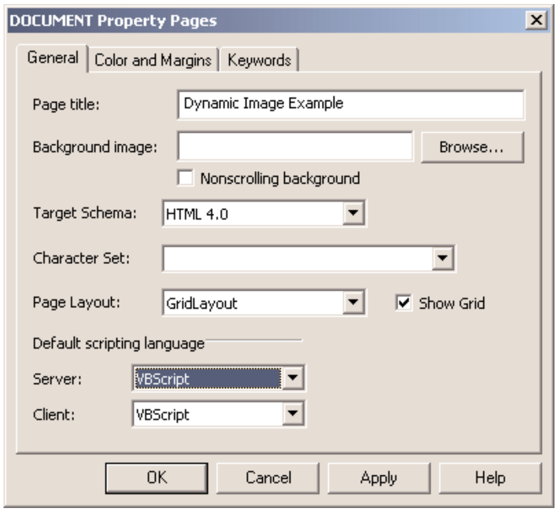
FIGURE 4.4

Solution Explorer pane after creating the `ch4` folder

TIP *If you usually prefer to edit the HTML directly, you can change the default by clicking Tools ➢ Options, and then selecting the HTML Designer item from the list of options. Change the Start Active Server Pages In option to Design View and then click OK.*

Right-click somewhere on the surface of the Web Form in the editing window, and select Properties from the context menu. You'll see the DOCUMENT Property Pages dialog (see Figure 4.5).

FIGURE 4.5
DOCUMENT Property Pages dialog



Page title: Dynamic Image Example
Background image:
☐ Nonscrolling background
Target Schema: HTML 4.0
Character Set:
Page Layout: GridLayout ☑ Show Grid
Default scripting language
Server: VBScript
Client: VBScript

Enter **Dynamic Image Example** in the Page Title field. Set the `targetSchema` property to Internet Explorer 5.0, and change the Page Layout setting to GridLayout. Hit OK when you are finished.

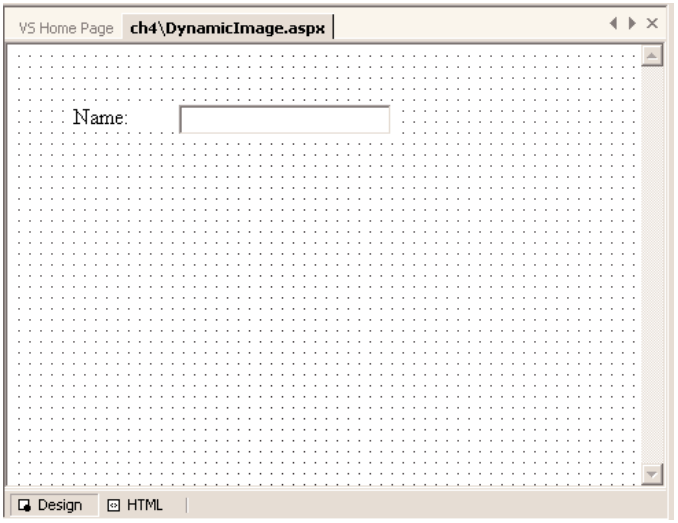
On the left side of your screen, you'll see a Toolbox tab. Move your mouse cursor over the tab; Visual Studio displays the Toolbox. Click the Web Forms bar, then click the Label Control item. Move your cursor back into the editing window and draw the Label control. You've just placed a Web Form label on the page. You should remember that Web Form controls are *not* the same as HTML controls, although they look identical; they have a different namespace from the equivalent HTML controls. Web Form Label controls and HTML Label controls (and most other controls that contain text) have a `Text` property like a VB.NET Windows Form Text control rather than a `Caption` property like a classic VB.NET Form Label control.

Next, drop a TextBox control next to the label. Your Web Form should look similar to Figure 4.6. That's it for page design—not elaborate, but functional. Next, you need to write a little code.

Step 3: Writing the Code behind the Page

Right-click the surface of the Web Form and select View Code from the context menu. If you're not very familiar with VB.NET, the code is somewhat intimidating, but don't worry, most of it is template code. The method you want to modify is the code for the `Page_Load` event.

FIGURE 4.6
The Dynamic-Image.aspx Web Form after placing controls



A Web Form executes the `Page_Load` event each time it's requested; however, the event contains code to differentiate between an HTTP `GET` request and a `POST` request. The correct .NET terminology is IsPostBack, meaning that the Web Form has been submitted back to the server. In other words, when IsPostBack is True, the user has already seen the Web Form page at least once.

In this sample page, you want to capture the text that the user enters into the text box, so your code will go into the bottom section of the If structure. For example, Listing 4.3 shows the code you need to add to the `Page_Load` event.

LISTING 4.3: THE *DYNAMICIMAGE PAGE_LOAD* EVENT CODE (*DYNAMICIMAGE.VB*)

```
' Generic VB.NET code omitted
Imports System.Drawing.Text
Private Sub Page_Load(ByVal Sender As System.Object, _
   ByVal e As System.EventArgs) Handles MyBase.Load
   If Not IsPostBack Then
      Response.Write("Page before posting<br>")
   Else
      Response.ContentType = "image/gif"
      getImage(TextBox1.Text).Save(Response.OutputStream, _
         System.Drawing.Imaging.ImageFormat.Gif)
      Response.End()
   End If
End Sub
```

I won't walk you through the entire code to create the image. But because this is some of the first VB.NET code listed in this book, I want to point out just a couple of things. First, the ability to dynamically create an image—any image—in memory simply wasn't available in classic ASP, or intrinsically in VB6 or earlier versions without extensive use of the Windows API. What you're seeing here is brand-new functionality. Second, while this function returns a Windows bitmap-formatted image, the `DynamicImage.aspx` Web Form returns a GIF-formatted image. In other words, VB.NET has the power to transform an image from a BMP to a GIF image. Again, this wasn't possible in earlier versions. Here's a function that creates a BMP file (yellow text on a black background) from the text entered by the user in the `DynamicImage.aspx` Web Form (see Listing 4.4).

LISTING 4.4: THE *GETIMAGE* FUNCTION (*DYNAMICIMAGE.VB*)

```
Public Function getImage(ByVal s As String) As Bitmap
   Dim b As Bitmap = New Bitmap(1, 1)

   'Create a Font object
   Dim aFont As Font = New Font("Times New Roman", 24, _
      System.Drawing.GraphicsUnit.Point)

   'Create a Graphics Class to measure the text width
   Dim aGraphic As Graphics = Graphics.FromImage(b)

   'Resize the bitmap
   b = New Bitmap( _
      CInt(aGraphic.MeasureString(s, aFont).Width), _
      CInt(aGraphic.MeasureString(s, aFont).Height))
   aGraphic = Graphics.FromImage(b)
   aGraphic.Clear(Color.Black)
   aGraphic.TextRenderingHint = TextRenderingHint.AntiAlias
   aGraphic.DrawString(s, aFont, _
      New SolidBrush(Color.Yellow), 0, 0)
   aGraphic.Flush()
   Return b
End Function
```

In the `Page_Load` method (see Listing 4.3), the page sets the `Response.ContentType` to `image/gif`, because the browser needs to know how to interpret the response. Next, it calls the `getImage` method and transforms the resulting BMP to a GIF file. Finally, it writes the binary stream of bytes containing the GIF file to the browser, which displays the image.

Web applications don't have a defined beginning and end—users can request any page in the application at any time. To test a specific page, you need to tell VS.NET which page it should run at startup. In this case, you want the `DynamicImage.aspx` page to appear when you start the program. Right-click the `DynamicImage.aspx` file in the Solution Explorer and select the Set As Start Page item from the pop-up menu.

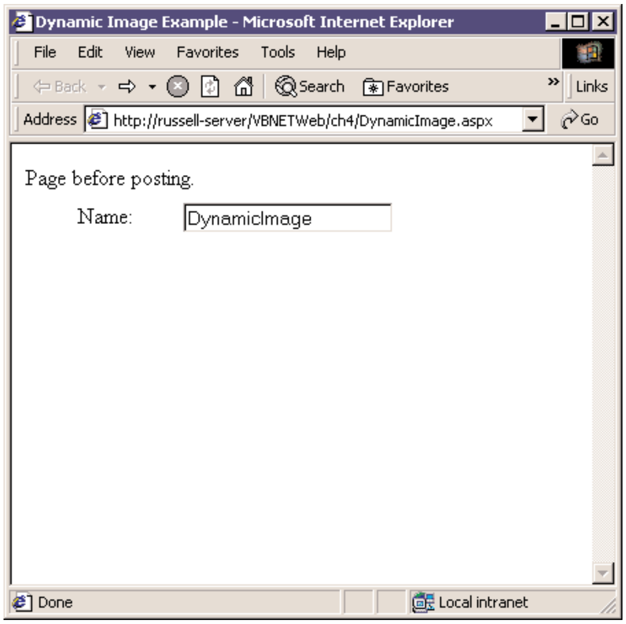
You can either build the project first, or you can simply tell VS.NET to launch the program, and it will build the project automatically. To build the project, use one of the "Build..." options on the Build menu. To begin running, you can click the Run icon on the toolbar, or press F5, or select Start from the Debug menu.

The first time you view the page, the `Page_Load` event fires, and you'll see the text "Page before posting." in your browser. In IE, HTML forms with a single `<asp:TextBox>` or HTML `<input>` control submit automatically when you press the Enter key.

When you submit the form, the browser requests the `DynamicImage.aspx` page again, this time with the `POST` request. The `Page_Load` routine fires again, but this time, IsPostback will have the value True (because it's a `POST` request), so the page performs the process to create an image.

Step 4: Viewing the Results in a Browser

Try it. Save the project and then press F5 to compile and run the project. The IDE opens up a new browser instance (see Figure 4.7).


Dynamic Image Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Links
Address http://russell-server/VBNETWeb/ch4/DynamicImage.aspx Go
Done Local intranet

Page before posting.

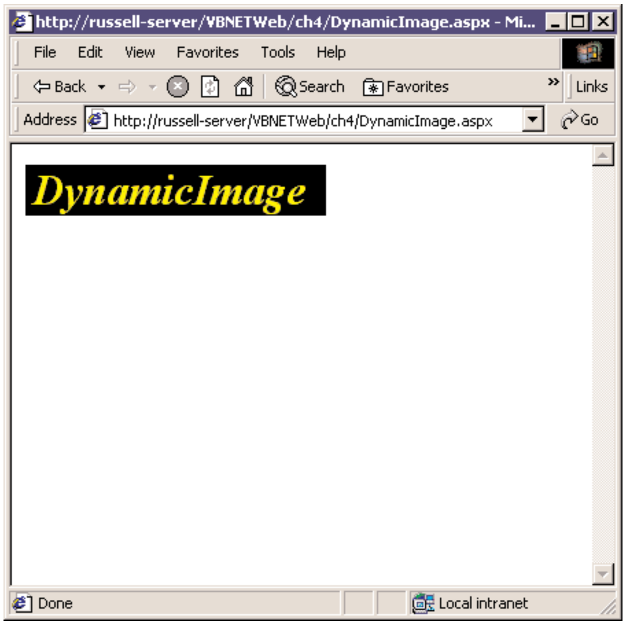
Name: DynamicImage

FIGURE 4.7
The `Dynamic-Image.aspx` Web Form before posting

Enter some text into the TextBox control and press Enter to submit the form. The server will respond with the text you entered in a GIF image sized appropriately to contain the text (see Figure 4.8).

Now, I don't know your level of experience with either VB6 or VB.NET, or with ASP.NET, but I can tell you—the first time I made this code run, I was seriously impressed. Not only is the code to create the bitmap only 10 (unwrapped) lines long, but there are no API calls, no handles, no special Declare statements, no memory management, no worries about memory leaks, and no DLLs to call or register. It just works. Now think of the hundreds of thousands of hours that people have spent doing exactly this kind of task for Web applications—drawing text in rectangles. I think this is a better way.

FIGURE 4.8
The `Dynamic-Image.aspx` Web Form after posting


http://russell-server/VBNETWeb/ch4/DynamicImage.aspx - Mi...
File Edit View Favorites Tools Help
Back Search Favorites Links
Address http://russell-server/VBNETWeb/ch4/DynamicImage.aspx Go
DynamicImage
Done
Local intranet

Summary

You've created a project and a Web Form, retrieved data from the client browser, and responded with custom code. At this point, you should begin to see how VS.NET has made the process of creating a VB.NET Web application very similar to the process of creating a standard Windows application—you create a project, create forms, drag and drop controls onto the forms, and write code to activate the form in a code window associated with, but not tightly bound to, that form. This loose binding is an improvement on ASP, because it facilitates code reuse. Just as you can create a generic Windows Form and use it repeatedly in multiple applications, you can do the same with Web Forms. You can reuse the user interface code of a Web Form by changing the code-behind class. Similarly, you can alter the look and layout of the user interface without recoding by changing the Web Form. Finally, you should realize that what you're really doing is using inheritance to customize generic classes to the needs of a specific application. The result is that you can now build a VB.NET Web application with a familiar set of tools and operations.

Whether you've been programming in VB6 or building ASP applications, the examples in this chapter should show you the greatly increased power of VB.NET and its tight integration with ASP.NET; but you've only begun to see the changes. In the next chapter, you'll explore Web Forms in much greater detail.