Beginning DotNetNuke 4.0 Web Site Creation

in VB 2005 with Visual Web Developer 2005 Express

From Novice to Professional

Nick Symmonds

Apress®

Beginning DotNetNuke 4.0 Web Site Creation in VB 2005 with Visual Web Developer 2005 Express From Novice to Professional

ISBN-13 (pbk): 978-1-59059-767-5

ISBN-10 (pbk): 1-59059-767-2

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

Lead Editor: Matthew Moodie
Technical Reviewer: Adriano Baglioni
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Elizabeth Seymour
Copy Edit Manager: Nicole Flores
Copy Editor: Heather Lang
Assistant Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor: Kinetic Publishing Services, LLC
Proofreader: Elizabeth Berry
Indexer: Broccoli Information Management
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

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

The source code for this book is available to readers at http://www.apress.com in the Source Code section.

CHAPTER 1

The Basics

This chapter will let you know what you need to prepare yourself for web page design. I'll tell you about the level of programming experience you need to get the most out of this book, and I'll let you know what you need to complete the projects in this book with respect to operating systems, memory, browsers, and so on.

Finally, I'll get into the development environments themselves. Yes, that was plural. In this book, you'll start with the Visual Basic 2005 Express (VB) IDE as a way of getting familiar with VB, the programming language used in this book. Later on, you'll graduate to the Visual Web Developer (VWD) 2005 Express IDE and combine it with DotNetNuke.

Note IDE is short for *integrated development environment*. The *integrated* part refers to the ability to edit, debug, and build a project all in one place. In fact, IDEs often allow you to check in and check out code from a source control database. If you ever work in collaboration with other programmers on the same project, you will need source control. For now, you can get away without it.

What You Need to Know

Here is where I need to be truthful about my level of delivery in this book. It is also where you need to know just what is expected of you. There are many things I will not cover in depth, simply because I expect that you are already familiar with them. Let's start with what you know.

Programming Experience

So how much programming experience have you had, anyway? Have you dabbled in Visual Basic? Have you created static web pages in HTML? Do you know what "VB" is? If the last three sentences totally rattle you, then this book is probably not for you.

While this is a book about beginning web page design, it's not a book about beginning programming for the totally uninitiated. You will be expected to know certain things, and

I will take you through mini-lessons on the things I think you may not know. Here is a list of the things you need to know about programming:

- What the different kinds of loops are
- How to create a function and how to call one
- How to use an editor
- Basic data flow and how to logically structure a program

It does not matter what programming language you are experienced in. It only matters that programming is not totally new to you. If you have spent a lot of time creating Visual Basic for Applications (VBA) routines for Excel or Word, you are well prepared for what is to come in this book. If you are a seasoned HTML and JavaScript programmer, you are even better prepared for this book.

Here is something else that I consider really important: you should have no fear of experimentation when it comes to programming. You should be comfortable around computers and be willing to experiment and learn. Often, the programming failures on the way to bug-free software can be more fun and instructive than if you hacked out perfect code to start with. I often find that failures because of bugs and lack of knowledge lead me down paths of learning that I never intended to explore in the first place.

Web Experience

Web experience can mean so many things. It can mean anything from reading the news sites to shopping on eBay or Amazon. To a hacker, it can even mean creating those dastardly pop-up ads that invade your web space.

The fact that you want to create web pages tells me that you have web experience. I bet you have a couple of browsers running—maybe Internet Explorer and Firefox. Here is a list of things that would be helpful as far as basic web knowledge goes:

- Knowing that there are many browsers out there that can show you the same web site
- Knowing that quite a few browsers are derived from the same basic browser engine
- Knowing key differences between browsers and why some people prefer one over another
- Knowing something about security in browsers and how to change it
- Knowing what a URL is
- Knowing what an IP address is and how it relates to DNS

- Knowing what HTML is
- Knowing what cookies are and how they are used
- Knowing how web pages are constructed
- Knowing what the Internet is and how you can use it effectively
- Knowing how to detect errors on a web page

Some of these things are rather advanced, I know. I did, after all, say they would be helpful, not required. During the course of this book, I will teach you about these things and more. By the end, you will be as well versed in browser lingo and manipulation as some of the best web designers. After all, isn't that why you are reading?

Basic Web Knowledge

Based on the preceding list, this section presents some things you need to know about the Internet and browsers. I will also tell you briefly how web pages are constructed and how they operate.

First of all, there are many browsers available to you. Any worthwhile one is free. There are more than just Internet Explorer and Netscape. However, these two are the most well known, because of the browser wars back in the late 1990s. (Sounds like an outer space conflict, doesn't it?) The most common browsers are Internet Explorer (IE), Netscape, Opera, and Firefox. As of the time of this writing, Firefox is gaining incredible ground on IE, and its uniqueness has finally triggered Microsoft to update IE. (IE V7.0 is scheduled to be released very soon.)

Next is the little known fact that many of these browsers are derived from the same basic engine. For instance, Netscape and Mozilla's Firefox are both derived from the same browser layout engine. This engine is called Gecko. I tell you this because you are much more likely to encounter similarities among Gecko-based browsers than between IE and Gecko-based browsers. In other words, Netscape is far more likely to work like Firefox than IE is. This is a great source of pain that VWD has resolved for you.

So what are some of the differences between browsers? Well, as someone who spends about 20 percent of each web project developing code that works on both major kinds of browsers (Gecko-based and IE), I can tell you there are some major differences and some minor ones. Some of the major ones follow:

- Some JavaScript errors kill IE but not Firefox.
- Some HTML tags are interpreted differently by IE and Firefox.
- IE and Firefox have totally different event models.

- IE does not fully support the implementation of cascading style sheets (CSS) 2.x; Firefox does.
- IE can run ActiveX programs (a security risk), and Firefox cannot (Firefox wins here).

The following are some of the minor differences you will see:

- Sometimes, different browsers position some tags differently.
- The order of HTML rendering is sometimes different in different browsers, which can make for strange appearances.
- Some style attributes that work in Firefox may not work in IE.
- Some things render faster in one browser than another.

The reason I tell you some of the differences among browsers is to prevent any undue hair loss. However, this may not always be something that can be helped.

There is a bright side to all this, though. Microsoft is very aware of all the browser differences, major and minor. VWD is designed to account for all these differences for you. It is very rare, indeed, that you have to discover which browser the client is running and adjust your code path to make allowances. I can guarantee you that in this book you will not have to worry about any of this. It is helpful, however, to keep this in the back of your mind.

Next, here are some web-related terms you should know, along with their definitions:

URL (uniform resource locator): This is what you type in the address bar at the top of your browser—you know, like `www.something.com`.

IP (internet protocol) address: Every computer or device in the world that is connected to the Internet gets an IP address. An IP address uniquely identifies the device on the Internet. Otherwise, how could you ever find it among the millions of devices on the Internet?

Router: This is a hardware device that steers information from one computer to another. If the router knows that the address you are looking for is in a particular area of the Internet, it does not broadcast your request everywhere. It directs it only to where it thinks you are looking. By the way, a router with DHCP and network address translation (NAT) has the ability to give out IP addresses and hide those addresses from the Internet as a whole. This means that there are several thousands of computers with the same IP address. No need to worry—the router takes care of this.

DNS (domain name system): This is the cool thing about the Internet that makes it accessible to the masses. A DNS server keeps a database of friendly names that match up with IP addresses. So if you have an IP address of, say, 10.44.33.126, the DNS allows you to type in `www.something.com` in the address bar and matches the friendly name with the IP address, so you get to where you want. Domain names are unique, as are IP addresses. Because of this, people will pay literally millions of dollars for a domain name, just because it is the same as their company name.

Cookies: These are small files that reside on your hard drive. Most every web site drops cookies on your machine when you visit it. These cookies contain information such as when you last visited a site, what page you were on, and so on. Cookies make it seem that a web site remembers you, but it's all an illusion. Cookies can also be used maliciously, such as in web site hijacking.

HTML (HyperText Markup Language): Basically, this is a set of elements delimited by tags in the form of `<tag>. . .</tag>`. Most of the time, these tags come in pairs, and the stuff in between is controlled by the tag. The tags are defined according to standards that are closely followed by all browsers (ha, ha)—at least, they should be. Reality, however, shows us that some tags are open to different interpretation by different browsers. Sometimes the differences are slight; sometimes they are major. What you need to know is that HTML is what makes a web page what it is. It tells the browser how to render the content.

Web server: This is a computer or set of computers that handles requests from browsers all over the Internet. Web servers return web pages and access databases when necessary. In your case, your computer is the web server, using Internet Information Services (IIS) to serve up pages in DotNetNuke.

Internet: I know, everyone knows what the Internet is, right? Did you know that at its root it is a collection of a dozen or so computers controlling DNS services and routing base traffic? Most people think the Internet is just there. Look up the history of the Internet sometime. It is very interesting.

Web Site Construction

Now that you have a basic understanding of the Web, it might be worthwhile to touch on how a web site works. Whether you program in VB, C#, Java, or ColdFusion, all web sites are essentially built in the same way.

First of all, the initial page of a web site is in a directory on a server somewhere. This directory could be several layers down the actual server's directory structure. If this were your web site, the web server would consider this the virtual root of your web site.

Under this root directory, you find subdirectories containing images (images are not contained in the web page but are referenced by it), other web pages, and server code.

This server code manages the business logic and database access for your web site. You also find a directory for the database if you have one. Figure 1-1 shows you a typical web site directory structure for a basic web site. This site was created using VWD.

Figure 1-1. *.NET web directory structure*

So here is essentially what happens when a web page is rendered on your machine:

1. The browser reads the incoming HTML text. As the text is read, it is parsed, and the screen is rendered.

2. The browser renders the HTML tags as they come in. There is no forward referencing of tags.

3. As image references are processed, the browser gets the images and displays them.

4. Events are fired, and various pieces of code are run.

I know this last step is rather nebulous, but this is where a good portion of the book resides.

Figure 1-2 shows a small web page. The HTML code behind this page follows it.

Figure 1-2. *Example of simple HTML code output*

Listing 1-1 shows the HTML code for this simple page.

Listing 1-1. *HTML code for two buttons in a table*

```
<table width="100%" border="1" bgcolor="green">
  <tr>
    <td width="5%"> </td>
    <td width="20% align="left">
      <input type="button" value="Press me" />
    </td>
    <td width="75% align="left">
      <input type="button" value="No, Press me" />
    </td>
  </tr>
  <tr>
    <td width="5%"> </td>
    <td colspan="2">
      <input type="checkbox" /> check here
    </td>
  </tr>
</table>
```

This is a simple table. The browser runs through the code from top to bottom and renders the tags as they appear. If you are new to HTML, this code may seem like Greek. Do not worry, as VWD can write most of this code for you. You just need to place your buttons and check boxes on your page visually, and VWD takes care of the rest.

HTML Primer

Let's look at the code from Listing 1-1 in a little detail. This small piece of code is pure HTML. It is an example of the most common way to place objects on the screen. In this case, the objects are two buttons and a single check box with some text. As you can see from the code and from Figure 1-2, I have used a table with rows and columns to create cells. These cells divide up the screen real estate into chunks. In these cells are the objects.

First, I have defined a table whose width is 100 percent of the width of the page. I have also defined the background color of this table to be green and to show the border. The vast majority of the time, you never show the border in any table. I've done it here for debugging purposes and to show you how it looks.

Next, I've defined two rows. These are marked off with `<tr>. . .</tr>` tags. The first row contains three cells (akin to columns) that are marked off with `<td>. . .</td>` tags. The first cell is 5 percent of the width of the table. It has only a space as its contents. This is defined by ` `, which means *nonbreaking space*. I've used this as a spacer. Using a `<td>` element as a spacer is very common. The second cell in the first row contains the

"Press me" button. Its width is 20 percent of the table width, and the button is left-aligned. The third cell in this row contains the other button. This cell is 75 percent of the table width and is also left-aligned.

Notice that the widths of all the cells make up 100 percent of the width of the table, which you should always try to maintain.

The second row contains only two cells. However, I need to keep the table balanced. In order to do this, I must span two of the cells in the first row with one of the cells in the second row. First, I make a spacer cell like I did in the first row. The next cell spans two columns as defined by the `<td>` attribute `colspan="2"`. This second cell contains the check box and the associated text.

Viewing Figure 1-2, you can see that the table is balanced, and the cells fill up the entire table.

I know that the explanation seems long-winded for a chunk of HTML that is so small. However, if you can understand this little piece of HTML and how it is rendered on your browser, you are a long way toward understanding how web pages really work. Now obviously, there are a ton more HTML tags, and each tag may have several attributes that define how it is rendered. I don't remember all this stuff, and I don't expect you to either. I use a certain percentage of tags in my work and know of most others. If I need in-depth information on how a tag is used or how to display something, I go to the Web. There are a great many web sites out there devoted to HTML tag explanations and examples.

When I am surfing, I keep an eye out for new ways to display things. If I see that someone has done something neat, I know that I can do it, too. It usually takes me only a few minutes to find an example or to figure it out myself. If you have a basic understanding and are willing to experiment, you can find out too.

■ **Tip** The HTML code for any page is viewable to the user. In IE, you can view the source code by choosing View ➤ Source from the menu at the top. The HTML code shows in a text editor. I do this all the time. You can get some neat pointers this way. Firefox has the same capability, through the menu command View ➤ Page Source.

When a user navigates your web site, she may click on menu items or links. What happens behind the scenes is that the web server calls up a new web page from one of the subdirectories under your web site. Essentially, all links are references to other pages either on your site or on another site.

One of the major things you need to be aware of during web site construction is the use of pictures and drawings. What follows is a small primer on images in web pages.

Images

Images on a page can be pictures or drawings or text. Text as a picture, you say? Well, consider the case in which a site developer wants to depict text in a certain font. Your machine is certain to contain many fonts—but what if he wants to use a unique font called, say, London Taxi? He can do one of two things: One, he can download this font to your machine and thereby proliferate this London Taxi font all over the world. However, this avenue has its pitfalls, one of which is that your browser may not allow a font to be downloaded to your machine. The better alternative is for him to write the text on his machine and take a picture of it. As far as your surfing goes, you don't see the difference between text and an image—it reads exactly the same. The big drawback to this method is that the text in the picture cannot be localized.

■ **Note** *Localization* is the process of externalizing all text and images from the code, so they may be translated for different cultures. .NET has a very structured method of localizing web pages into many languages, so you can easily switch between them.

Anyway, back to images. As I said before, when you construct a web page, the image is not part of the page itself. Instead, the image gets rendered at the place inside your page where the image tag is located.

There are different kinds of images available that can be used. They each have different qualities depending on the attributes you need. Table 1-1 explains the common ones.

Table 1-1. *Image Types and Their Pros and Cons*

Image Type	Extension	Pros	Cons
Bitmap	`.bmp`	Format is universal	File size can be large; does not support transparency
JPEG	`.jpg`	File size is small; supports 24-bit color	Uses lossy compression, in which some data is lost
TIFF	`.tif`	Uses lossless compression, so all image data is retained	File size is large
GIF	`.gif`	Uses lossless compression; can contain transparency	Uses only 256 colors; sometimes involves patent issues
Animated GIF	`.gif`	Animates with no extra programming by you	Same as GIF

These are the major image formats. Personally, I prefer JPEG and GIF formats. I like JPEG for photographs, and I like GIF for all other images. You will not find a graphical browser that does not support these two.

Got a camera? I bet, since you want to create a good web site, that you have a really nice digital camera with over 5 megapixels that shows images in stunning color. Great for the web site, eh? Wrong!

How often have you been surfing the web and come across a web site loaded with images, and your computer grinds to a halt? Unless the site was really important to you, chances are you left immediately. In fact, studies on load testing have shown that the average person's tolerance for waiting for a web site to render is 8 seconds. I have been to web pages that take almost a minute to load at high speed, because of many high-resolution images being downloaded to my machine.

If you are taking pictures to include on your web site, I suggest that you take them at a low resolution. Think of the size of the picture and how it will be displayed on your web page. Most likely, it will only take up a small space. If this is the case, "dumb down" your camera to 640×480 resolution and then take the picture. The resulting file size of this picture will be orders of magnitude smaller than a high-quality color-dense photo. Smaller file size means faster rendering speed on your client's browser. Remember the 8-second rule for page rendering (not to be confused with the 5-second rule for food dropped on the floor.)

Figure 1-3 shows a low-resolution picture of my bicycle. I originally took the picture at two resolutions: a high resolution, which resulted in a file size of 355 KB, and a low resolution, which resulted in a file size of just 48 KB. The JPEG files themselves are available from this book's download page at `www.apress.com`. Compare them yourself.

Figure 1-3. *Picture of a bike taken at a low resolution of 640×480 pixels*

Most pictures do not take up the whole page. If they only take a small portion of the page, use a lower resolution. The photos will still look fine.

How a Web Page Works

I've told you a little about how a web page is constructed. Now I will tell you a little about how a web page works. There are several attributes of a web page that make it different from your classic Windows application (you know, applications like Excel, Word, Photoshop, etc.), and I'll discuss these in the following sections.

State

Web pages are stateless. The programs I mentioned previously are stateful. *Stateful* means that the program remembers what you are doing. The program remembers what is on your display and where the data resides. The program knows at all times what you are doing with the data. All Windows programs are stateful.

Stateless refers to the fact that once the program sends the data to the display, it suddenly forgets all about you. In fact, as far as the program is concerned, it never happened. The program has no idea what data it sent and has no idea what you are doing with the data. Web programs are stateless. They preserve state for a fraction of a second at best—just long enough to get the data to you.

Why are web programs stateless? It is the nature of the Internet. There is no persistent connection between a web server and a web browser client. In order for a web server to scale to hundreds of thousands of connections per day, there can't be a persistent connection. No server could handle this. Also, the Internet is built to withstand breaks in communication lines. If a line goes down or a router goes bad, the connection can be rerouted via another path. The Internet is error-resistant and self-healing to an extent. There is also so much traffic over the Net that holding a connection open between two computers would take up too much precious bandwidth. The Internet's statelessness allows you to surf to a site and then go have dinner with no harm done and no bandwidth taken.

But how does a site remember you? How are you able to shop online and have the cart constantly updated? This is all an illusion and a great deal of programming.

There are ways to achieve the personalized experience you know from modern-day web sites. I am sure you have shopped online, used a shopping cart, and so on. You've probably also come back to some of these sites and they "remember" you. Cool, huh?

The most common way to simulate state is though a session ID. Once you hit a web server and request a page, the server sends the page with a session ID to the browser. Each time you make another call to the web server, the browser sends this session ID back to the server with your request. This session ID is used to track you and what you are doing at the moment. It is also used as a security token to allow you to see some pages and not others, depending upon your login.

Session IDs are often fleeting. For example, with some sites, when you log on and get a session ID, the web server starts a countdown timer. If you do not get back to the site within a certain timeout period, the session ID expires and the web server logs you out automatically. You see this a lot with e-commerce sites.

■ **Note** You should note that in traditional web site development, state management is quite the programming task. However, by using ASP.NET, you are covered. Microsoft takes all the state management stuff out of the programmer's hands and manages it automatically for you. There are still ways, however, to manage state yourself if needed.

Managing session state is key to providing a rich user experience with your web site.

Events

OK, let's say you're on a web site, and you choose some value from a drop-down list. Or perhaps you move your mouse over a word and a picture pops up or changes. What is going on in these scenarios?

If you are a traditional Windows programmer, you know that whenever you interact with a control like a drop-down box or button, an event is fired. The Windows operating system allows you to catch the event and write code to respond to that event. The same thing can happen in a web application.

There is a difference, though, between how events are handled in web pages and how they're handled in Windows programs. For one thing, with web pages, you get to decide where the event is handled. This is really important for ASP.NET developers like you. ASP.NET allows you to handle the event in the client's browser or back at the server. Figure 1-4 shows a diagram of browser-based event handling (also known as client-side event handling).

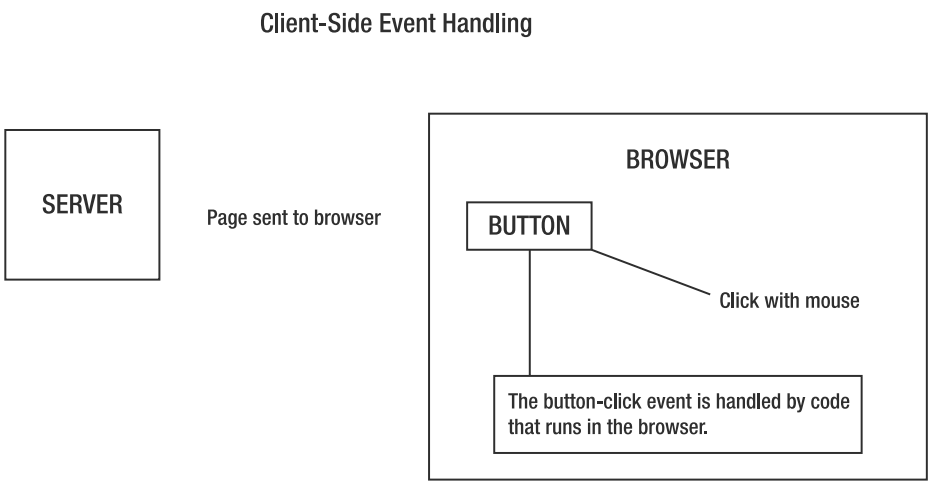

Figure 1-4. *Client-side event handling*

Figure 1-5 shows a diagram depicting events as they are handled on the server.

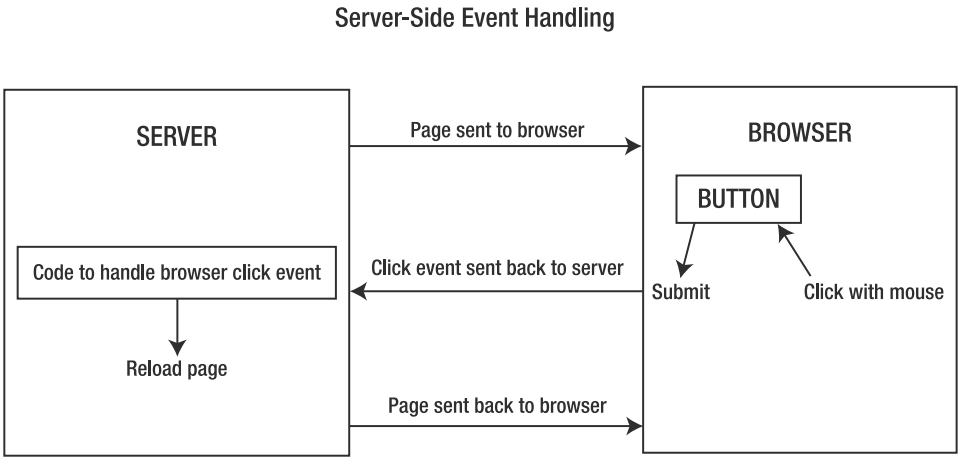

Figure 1-5. *Server-side event handling*

Simply by looking at the number of arrows in the respective diagrams, you can see that the client-side event handling is far simpler than the server-side event handling. If you choose server-side event handling, several things take place before all is said and done:

- The browser detects the click and decides what to do with it.
- The browser submits the request back to the server. This is called a *postback*.
- The server runs code that handles the event.
- The server renders the page again with any changed data for any fields on the page.
- The server sends the page back to the browser.
- The browser renders the page.

While this often happens in the blink of an eye, it is a lot of work. More to the point, it takes a complete round-trip to handle any event. For example, if you caught a change mouseover event for several pictures on a page, the round-trips to the server could get annoying to the customer. You see, every time you submit the page to the server, it has to render the whole page again and send it back. This results in the familiar flicker you often see on web pages. If you have a dense web page with lots of pictures, these events can really slow down the interaction speed.

Therefore, it is best to handle simple events on the browser when possible. It makes for a quicker and more realistic experience for the user. However, do not be afraid of using server-side events. ASP.NET makes heavy use of server-side events, and they can be very handy and powerful to program.

■ **Note** An advanced method of interaction between the client and the server—called Ajax—is becoming very popular these days. It is a method in which the client can make requests of a server for data without submitting the page. The server can send back data, and the client can then update a single field in the page. This is very fast and requires no rerendering of the page (more on this technique much later in the book).

Postbacks

I mentioned in the previous section that server-side event handling is done via a postback or page submittal. Submittals are how the browser sends data back to the server.

You can design a page with several controls. One special control is called a submit button. When this button is pressed in the browser, the page is automatically submitted back to the server. The server can read all the controls on the page and the values of those controls. Once the server has read the control values, it can save the data to a database and perform some business logic based on the values.

When the server is done processing the posted page, it sends back the same page (with the same or different values) or a different page. Remember this, though: even if nothing changes on the page, the server must rebuild the entire page, values and all, before sending it as HTML back to the browser. While this process may seem wasteful, it has worked for years and is the standard way of doing things.

While postbacks seem inefficient, I should say here that web servers such as IIS are very efficient at rendering HTML for web pages. Such web servers use all kinds of memory caching, which speeds up processing to the point that the user won't even notice the flicker, at least on lightweight pages.

What You Need to Have

I have so far explained in general terms what you need to know about programming and web sites to continue with the book. This section lets you know what you need in terms of equipment and software.

Hardware

Pretty much any modern-day computer will do. You will need a decent amount of hard drive space to hold the two Express IDEs and any web sites you create. About 20 GB of free space will be OK.

You will need a high-speed Internet connection. A telephone modem will not do, but basic DSL or a cable modem will be fine. You will want to eventually access your computer from other computers to see if your web site works.

Software

First on the software list is the operating system. I recommend the latest from Microsoft, which at the time of this book's publication is Windows Server 2003. I recommend this operating system, because it allows you to upgrade to the full Visual Studio (VS) 2005 edition (complete with source control, etc.), which you may want to do in the future. Some parts of VS 2005 require Windows Server 2003 as the operating system. However, if you don't have it, you can get away with Windows 2000 Professional or XP Professional just fine for this book.

■ **Note** If you're using Windows XP Home Edition, you may want to upgrade to XP Professional with Service Pack 2 (SP2). XP Professional allows you to run IIS, which allows you to better test your web application. However, XP Home will suffice, because VWD has its own web server. Chapter 3 deals with installing DotNetNuke on XP Home and XP Professional.

You will need the latest .NET Framework that works with Visual Studio 2005. As you load the Express IDEs, it will detect if you have the .NET 2.0 Framework. If not, it will be installed automatically.

If you have an existing framework from version 1.x of .NET on your machine, don't worry. Both frameworks can reside together and can even be run together. There is nothing stopping you from working on .NET 1.x stuff and .NET 2.0 stuff at the same time. Being able to run two versions of the same code at the same time is one part of eliminating DLL hell (explained in Chapter 2). Those of you who are programmers can rejoice. Those who have never experienced DLL hell don't need to worry. Just know that you have avoided something from Microsoft that has plagued us developers since the beginning of time (at least since the 1980s, anyway).

Programming Environments

You will be loading two programming environments on your machine: Visual Basic 2005 Express and Visual Web Developer 2005 Express.

I'm having you load VB Express to help with Chapter 4. VB is the language of choice in this book, and Chapter 4 goes over the basics of VB and how to create Windows programs. This discussion has a twofold purpose: first, to get you familiar with VB and Windows programming, and second, to give you a reference when building web pages. You will find

that building a web page that reflects a Windows program is an eye-opening experience. Knowing how a Windows program works gives a great appreciation for the strengths and weaknesses of web development.

Visual Basic As a Language

So, why is VB the language of choice for this book? I list some reasons for you here, and then I will them explain in a little more detail:

- VB has been around in different forms for a long time—it is a very mature language.
- VB for .NET allows you to migrate easily from VB6. The syntax is close enough to make you feel comfortable from the start.
- DotNetNuke is written in VB.
- VB is a very efficient language.

Before discussing the points in the list, let me give you a bit of history. The first high-level language was the C language. In my beginning programming days, I used C extensively. It was the language of choice for systems programming, and it was platform independent. Those days, "platform independent" meant Unix or DOS 3.3. (Can you guess my age yet?) I programmed both firmware and software in C. I was able to develop code on my PC and run it on the Unix box.

After C came C++. It was the turbo-charged, object-oriented version of C. During the object-oriented phase in programming, there was also Delphi (object-oriented Pascal), Eiffel (not seen much these days), and Java (seen everywhere these days).

I turned to VB at version 5.0. I immediately liked its ease of use as compared to C++. I especially liked the fact that I no longer had to deal with pointers, because like many other programmers, I found multiple levels of indirection frustrating.

Visual Basic 4.0 came out in August 1995. This was a major release for VB, in that it became much more of an object-oriented language. VB has done so much for the programmer in terms of prewritten code (in the VB library) that programmers are able to write multitier Windows programs with almost no effort. Visual Basic hides from the programmer many of the complex operations that have to be done manually in C++. Among the things that VB is especially good at hiding are the inner workings of the component object model (COM) and the distributed component object model (DCOM). Several years ago, COM was the in thing, and either you were an expert in ATL or you used VB. These days COM is vilified (for good reason), and .NET uses other methods to connect among different tiers of a program.

Visual Basic version 5 (1997) and version 6 (1998) continued to enhance the programmer's experience with better IDEs and the ability to easily create program tiers that communicate remotely with each other using COM. As I said, COM was easy, but it opened up a whole can of worms when it came to playing nice with other programs.

Visual Basic gained serious traction over the past decade, not only because it was far easier for nonprogrammers to learn but because VB was the basis for extending the Microsoft Office Suite of programs. Visual Basic for Applications (VBA) is used heavily by all manner of office workers to enhance Microsoft Word or Excel.

In 2001 Microsoft released the first version of Visual Basic for .NET. While this next version was syntactically similar to VB6, it was able to utilize the full power of the .NET framework.

Since VB, C#, and any other .NET language get translated to the same intermediate code, there is no difference in the power of one language over the other. VB can perform the same operations that C# can in the same time. The power of DNN resides in the fact that DotNetNuke is written in VB.

ASP.NET for Web Development

The second environment you will load is Visual Web Developer 2005 Express Edition. This is where the rubber meets the road, so to speak. VWD is your gateway to creating web pages and sites that achieve your goals. VWD is not just an IDE—it is an environment in which you can use ASP.NET, HTML, JavaScript, C#, and VB. For this book, the choices of programming languages are VB for business logic and ASP.NET and HTML for presentation.

This book is basically about ASP.NET. VB and DotNetNuke are just ways to make developing in ASP.NET easier. If you are not very familiar with ASP.NET, here is the lowdown.

There have always been two major camps for enterprise web development. One comes from Sun Microsystems, and the other, of course, is from Microsoft.

Sun gives us the Java/JSP/Servlet/JavaScript development combination. Microsoft gives us the ASP/VB/VBScript development combination. ASP and JSP both give us dynamic content in web pages.

The basic difference comes down to price of development tools and price of the web server. Java-based web pages can be developed for free and run on a free Apache web server. ASP-based web pages have better (but not free) development tools. (However, ASP can be written using just Notepad.) ASP also requires that you use IIS, which is not free.

Along comes ASP.NET from Microsoft with killer development tools that are free (via VWD) and a web server that can be used to test your web page (also free, via VWD). The really nice thing about ASP.NET, though, is that it has fixed all the shortcomings of Java/JSP development and classic ASP development.

The DotNetNuke Difference

The last thing you need to load onto your machine is DotNetNuke (DNN). DNN is not an IDE like VWD. Rather, it is a framework of wizards, templates, and code that allows you to create professional web pages in a flash. Working with DNN and VWD gives you all the tools you need for quite a while.

DNN is not a language, and it's not a development environment either. In fact, DNN allows you to program ASP.NET pages in C#, VB .NET, J#, and even COBOL.NET (if you are feeling masochistic).

DNN is a framework that allows you to create web sites faster and with more consistency than otherwise. DNN is very popular and is getting more so every day.

So what does DNN bring to the table? While VWD allows you to do everything you need to create a web site, DNN is a framework that manages it all for you. A web site is not just a page that is shown to a surfer. It is a collection of database tables, back-end logic, administrative duties, security concerns, and so on.

DNN allows you to do site hosting if you like. It allows you to manage content and site membership. Although you can do all this development yourself using VWD, using the help of DNN tools makes it much faster and easier.

Web Server

So, where do web pages come from, you ask? I think you are old enough now to know the truth: they come from a web server. A web server is often thought of as a nebulous high-powered machine that resides somewhere in California. It could be, but in your case it's not. For all the projects in this book, the web server is your development machine. Perhaps, after you've made your web site, you'll want to upload it to one of these high-powered servers. I will tell you how to do this near the end of the book, in Chapter 10.

There are two web servers you need to worry about. The first is a small, test web server that comes with VWD. It is a single-use server that is meant only for displaying web pages as you develop them on your machine. It cannot be used by any other machine. This is cool, as it does not have to be configured, and there is no worrying about how it works. It just does.

The second web server you need to be concerned about is IIS. (As mentioned in the beginning of the chapter, IIS stands for Internet Information Services.) It is Microsoft's web server. It comes on every machine that has Windows 2000 Server or later. However, it needs to be set up and configured to run. You will use IIS to test your web sites' deployment and to do some load testing (for cases when several PCs hit your site at the same time). I will cover IIS in detail in Chapter 3.

Configuring the Browser

For the web pages that you create and display in your browser, you will need to be able to view what you've rendered, run any JavaScript code, and debug any strange behavior. In order to do all this, you will need to configure your browser.

Browser Security

Browser security is where the last decade of security worry has been directed. Everyone these days gets online and browses sites that they shouldn't. Most of the time, this is through no fault of their own, but they are drawn there by mislabeled sites and false advertising.

The browser is a neat tool that can unfortunately let in nasty viruses and other bad surprises—thus the need for browser security.

Here is a list of things you can do to your browser to harden it against malicious attacks:

- Don't allow any cookies.
- Don't allow any JavaScript to run.
- Don't allow any VBScript to run.
- Don't allow any ActiveX components to run.
- Don't allow any ActiveX controls to be downloaded.
- Don't allow fonts to be downloaded.
- Enable the pop-up blocker.
- Severely restrict Java.

This is just the major stuff. There are a ton of other settings you can enable to make your browser safe—and practically unusable to both you and any outside force. That is the problem with security. How much is enough so that you can still use your browser and not be hamstrung by it? Let's look at some of the settings that affect security in a surfing sense, as well as from a web development perspective.

Cookies

These are files that the browser lets you put on the client's machine. Think of the browser as a window from the outside world into your machine. Yes, you can see out, but others can also see in. There are two things you do not want the outside world to do on your machine: one is to run code that can affect your machine; the other is to drop a file on your machine.

I will cover the code part later. The file part is solved by cookies. Cookies are files that are created and destroyed by the browser itself. You can write some scripting code that runs on a client's browser that tells the browser to create, destroy, or read a cookie. So while people may not be able to send unsolicited files to your machine though the browser, they can tell the browser to drop a cookie.

So what are cookies used for? They are used mainly to keep track of you. If you surf to a web site, often it drops a cookie noting that you have been there. The contents of a cookie might be the last page you visited at that site or the last date of the last movie you rented from that site. A cookie can also contain a username and password. Some sites use cookies to log you back in when you return to a site.

Remember I talked a little about state before? A cookie is essentially a way to remember state. Recall that the web server sends a page to you and then forgets all about you.

Since the server cannot remember that you were ever there, the next best thing is to have your computer remember that you were there. As you interact with a web site, the cookie is sent back and forth as a memory device.

The bad thing about cookies is that they can be spoofed and stolen. Web sites trust the cookies that they leave. If another web site changes them, the web site that left them will not know that some information in them is false.

So the end result of all this worry is that some people turn off cookies. This effectively gives your computer amnesia every time it visits the same site. Now that there are so many sites that are tailored to the user, it can be really annoying to have to constantly type in the same information every time you visit. However, this is life, and you will find that some clients' machines have cookies disabled.

For this book, however, you need to allow cookies. To do this in IE 6.0, choose Tools ➤ Internet Options, click the Privacy tab, and set the slider to the Medium setting (as shown in Figure 1-6).

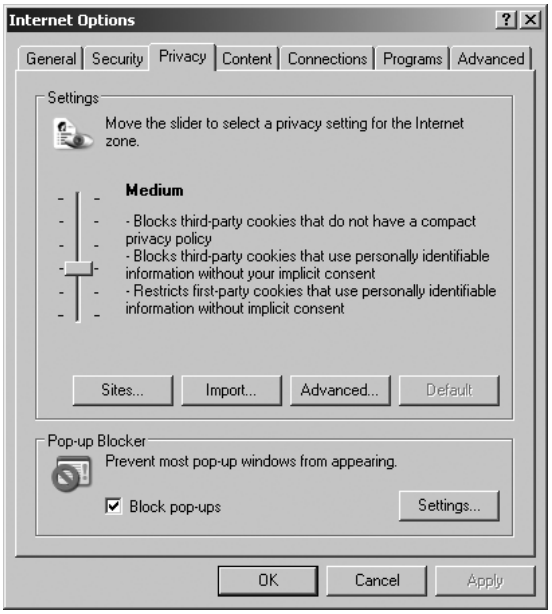

Figure 1-6. *Allowing cookies in IE 6.0*

In fact, all of the security options mentioned here can be activated in IE 6.0 through the menu option Tools ➤ Internet Options. Firefox can also disable or enable cookies. You can do this by navigating to Tools ➤ Options ➤ Privacy ➤ cookies.

ActiveX Components

ActiveX components are controls that are able to make full use of your computer's resources. This means, for example, that they can use the graphics device interface (GDI) to perform very complicated drawing. They can use the file system to read and write files on your hard drive. They can use the operating system to control just about anything on your machine. Note that ActiveX components are available in IE but not in Firefox.

In effect, an ActiveX component is a control that runs just like any other executable on your machine.

While ActiveX controls give a richer user experience to your web page, I suggest that you not only don't use them but that you turn them off. Most web sites and web clients these days do not use anything that must be downloaded to the client's computer. Most businesses do not allow it. To turn them off in IE 6.0, go to Tools ➤ Internet Options ➤ Security ➤ Custom Level (as show in Figure 1-7). You can disable aspects of ActiveX controls here.

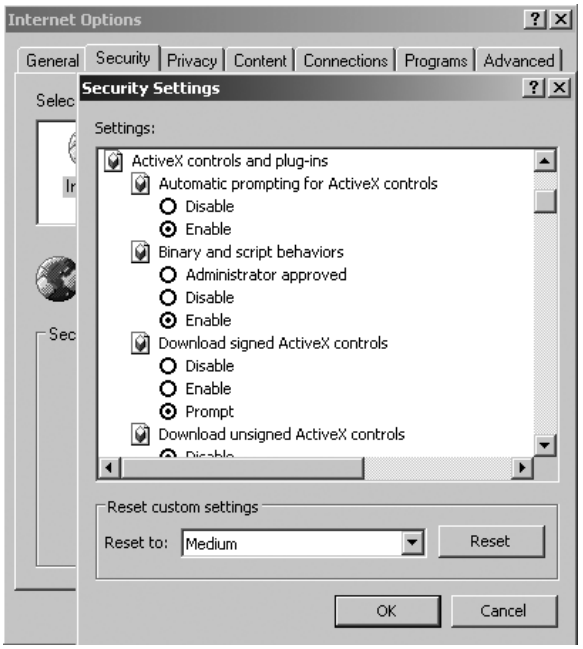

Figure 1-7. *Security for ActiveX controls*

By the way, do you know that there is a Java analog to the ActiveX component? It is called an applet. An *applet* is a small Java program that is downloaded to your machine and acts as a helper for the web page. It serves the same purpose as an ActiveX component. Most businesses do not allow applets either.

Scripting

Scripting is something that brought web pages out of the dark ages and into the full user experience you know today. *Scripts* are small functions that run inside your browser. These scripts control how the HTML controls act and react to user input. They allow the dynamic interaction that you get with most web sites. Scripts are different from ActiveX controls or applets, as they are controlled by your browser and run in a sandbox.

You can certainly interact with the user on your web site without using scripting, but it's slower and harder. For instance, many HTML controls, such as text boxes or buttons, have events that you can catch and respond to. There are two ways to do this.

The first way is to submit the page to the server, figure out what event was fired, react to the event, and return the page in whatever new state it needs to be in. This can be pretty fast on a small page with few controls, but it can be pretty slow on a very dense page with many HTML elements on it.

The second way is to program event handlers though JavaScript or VBScript. This allows the client browser to handle the event with no posting to the server. No round-trip means no delay. There are many things you can do using client-side scripting that does not need any server intervention. I'll explain this in detail later in Chapter 12.

ASP.NET has the unique ability to use either client-side scripting or server-side code for event handling. You have the ability to redirect event handling at will. The best option is to use client-side scripting whenever possible, and server-side code when serious business logic is involved. You can even detect when the user has scripting disabled, and then resort to server-side code.

Most businesses allow client-side scripting, so you need to turn it on. You need to be able to work on computers that have scripting on and off. Figure 1-8 shows the security setting for this.

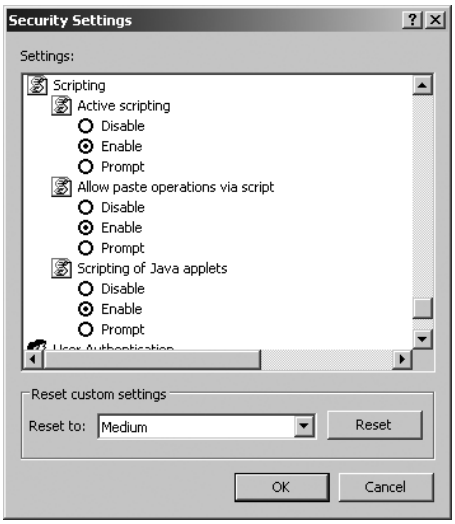

Figure 1-8. *Enabling active scripting*

Debugging

There is one last thing you need to do before starting out in web programming. You've just enabled web scripting; now you need to be able to debug it. VWD works just fine for debugging VB code and any other server-side code—however, it cannot debug code that runs in the browser. IE can. You can allow client-side JavaScript debugging with a tool called Microsoft Script Editor (MSE) that comes with Microsoft FrontPage 2003. Figure 1-9 shows how to allow debugging in IE.

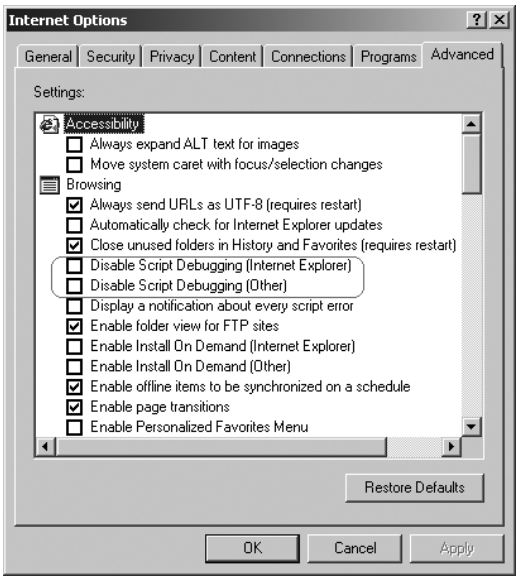

Figure 1-9. *Allowing client-side script debugging*

Note that you need to *uncheck* these options in the Advanced tab of the Internet Options menu (this was confusing to me at first).

If you do not have FrontPage loaded, there are other debugging techniques that I will go over with you in Chapter 12.

Summary

This chapter has covered the basics of what you need in order to effectively use this book. By "basics," I mean not only the physical aspects but the conceptual ones as well.

You need to know something about web pages and maybe a little programming. Mostly, though, you just need to be willing to experiment with writing code. If you are willing to try out new things, you can go far in creating web pages.

This chapter has also covered the hardware and software necessary for the projects in this book. The hardware is something you most likely already have, and all of the software you need is free.

Chapter 2 will explain why the combination of VWD and DotNetNuke is the choice for you to create professional web pages in a short amount of time.