Pro ASP.NET 2.0 in VB 2005

Laurence Moroney
Matthew MacDonald (Ed.)

Apress®

Pro ASP.NET 2.0 in VB 2005

Copyright © 2006 by Laurence Moroney, Matthew MacDonald (Ed.), K. Scott Allen, James Avery, Russ Basiura, Mike Batongbacal, Marco Bellinaso, Matt Butler, Andreas Eide, Daniel Cazzulino, Michael Clark, Richard Conway, Robert Eisenberg, Brady Gaster, James Greenwood, Kevin Hoffman, Erik Johansson, Angelo Kastroulis, Dan Kent, Sitaraman Lakshminarayanan, Don Lee, Christopher Miller, Matt Milner, Jan Narkiewicz, Matt Odhner, Ryan O'Keefe, Andrew Reid, Matthew Reynolds, Enrico Sabbadin, Bill Sempf, Doug Seven, Srinivasa Sivkumar, Thiru Thangarathinam, Doug Thews

ISBN-13 (pbk): 978-1-59059-563-3
ISBN-10 (pbk): 1-59059-563-7

Library of Congress Cataloging-in-Publication data is available upon request.

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

Lead Editor: Tony Davis
Technical Reviewer: Andy Olsen
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Richard Dal Porto
Copy Edit Manager: Nicole LeClerc
Copy Editor: Kim Wimpsett
Assistant Production Director: Kari Brooks-Copony
Production Editor: Linda Marousek
Compositor and Artist: Kinetic Publishing Services, LLC
Proofreaders: Linda Seifert and Nancy Sixsmith
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. You will need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 1

Introducing ASP.NET

When Microsoft created .NET, it wasn't just dreaming about the future—it was also worrying about the headaches and limitations of the current generation of web development technologies. Before you get started with ASP.NET 2.0, it helps to take a step back and consider these problems. You'll then understand the solution that .NET offers.

In this chapter you'll consider the history of web development leading up to ASP.NET, take a whirlwind tour of the most significant features of .NET, and preview the core changes in ASP.NET 2.0. If you're new to ASP.NET, this chapter will quickly get you up to speed. On the other hand, if you're a seasoned .NET developer, you have two choices. Your first option is to read this chapter for a brisk review of where we are today. Alternatively, you can skip to the section "ASP.NET 2.0: The Story Continues" to preview what ASP.NET 2.0 has in store.

The Evolution of Web Development

More than ten years ago, Tim Berners-Lee performed the first transmission across HTTP (Hypertext Transfer Protocol). Since then, HTTP has become exponentially more popular, expanding beyond a small group of computer-science visionaries to the personal and business sectors. Today, it's almost a household term.

When HTTP was first established, developers faced the challenge of designing applications that could discover and interact with each other. To help meet these challenges, standards such as HTML (Hypertext Markup Language) and XML (Extensible Markup Language) were created. HTML established a simple language that could describe how to display rich documents on virtually any computer platform. XML created a set of rules for building platform-neutral data formats that different applications could use to exchange information. These standards guaranteed that the Web could be used by anyone, located anywhere, using any type of computing system.

At the same time, software vendors faced their own challenges. They needed to develop not only language and programming tools that could integrate with the Web but also entire frameworks that would allow developers to architect, develop, and deploy these applications easily. Major software vendors including IBM, Sun Microsystems, and Microsoft rushed to meet this need with a host of products.

ASP.NET 1.0 opened a new chapter in this ongoing arms race. With .NET, Microsoft created an integrated suite of components that combines the building blocks of the Web—markup languages and HTTP—with proven object-oriented methodology.

The Development World Before ASP.NET

Older technologies for server-based web applications rely on scripting languages or proprietary tagging conventions. Most of these web development models just provide clumsy hooks that allow you to trigger applications or run components on the server. They don't provide a modern, integrated framework for web programming.

Overall, most of the web development frameworks that were created before ASP.NET fall into one of two categories:

- Scripts that are interpreted by a server-side resource
- Separate, tiny applications that are executed by server-side calls

Classic ASP (Active Server Pages, the version of ASP that predates ASP.NET) and ColdFusion fall into the first category. You, the developer, are responsible for creating a script file that contains embedded code. The script file is examined by another component, which alternates between rendering ordinary HTML and executing your embedded code. If you've created ASP applications before, you probably know that scripted applications usually execute at a much slower rate than compiled applications. Additionally, scripted platforms introduce other problems, such as the lack of ability to control security settings and inefficient resource usage.

The second approach—used widely by, for example, Perl over CGI (Common Gateway Interface)—yields an entirely different set of problems. In these frameworks, the web server launches a separate application to handle the client's request. That application executes its code and dynamically creates the HTML that should be sent back to the client. Though these applications execute faster than their scripted counterparts, they tend to require much more memory. The key problem with this sort of approach is that the web server needs to create a separate instance of the application for each client request. This model makes these applications much less scalable in environments with large numbers of simultaneous users, unless you code carefully. This type of application can also be quite difficult to write, debug, and integrate with other components.

ASP.NET is far more than a simple evolution of either type of application. Instead, it breaks the trend with a whole new development model. The difference is that ASP.NET is deeply integrated with its underlying framework. ASP.NET is *not* an extension or modification to the .NET Framework with loosely coupled hooks into the functionality it provides. Instead, ASP.NET is a portion of the .NET Framework that's managed by the .NET runtime. In essence, ASP.NET blurs the line between *application* development and *web* development by extending the tools and technologies previously monopolized by desktop developers into the web development world.

What's Wrong with Classic ASP?

If you've programmed only with classic ASP before, you might wonder why Microsoft changed everything with ASP.NET. Learning a whole new framework isn't trivial, and .NET introduces a slew of concepts and can pose some serious stumbling blocks.

Overall, classic ASP is a solid tool for developing web applications using Microsoft technologies. However, as with most development models, ASP solves some problems but also raises a few of its own. The following sections outline these problems.

Spaghetti Code

If you've created applications with ASP, you've probably seen lengthy pages that contain server-side script code intermingled with HTML. Consider the following example, which fills an HTML drop-down list with the results of a database query to get author details from the Pubs database in SQL Server:

```
<%
  Set dbConn = Server.CreateObject("ADODB.Connection")
  Set rs = Server.CreateObject("ADODB.Recordset")
  dbConn.Open "PROVIDER=SQLOLEDB;DATA SOURCE=(local);
            DATABASE=Pubs;User=sa;Password=sa"
%>

<select name="cboAuthors">
  <%
    rs.Open "SELECT * FROM Authors", dbConn, 3, 3
    Do While Not rs.EOF
  %>
  <option value="<%=rs("au_id")%>"><%=rs("au_lname") & ", " &
    rs("au_fname")%></option>
  <%
    rs.MoveNext
    Loop
  %>
</select>
```

This example needs an unimpressive 16 lines of code to generate one simple HTML control. But what's worse is the way this style of coding diminishes application performance because it mingles HTML and script. When this page is processed by the ASP ISAPI (Internet Server Application Programming Interface) extension that runs on the web server, the scripting engine needs to switch on and off multiple times just to handle this single request. This increases the amount of time needed to process the whole page and send it to the client.

Furthermore, web pages written in this style can easily grow to unmanageable lengths. If you add your own custom COM components to the puzzle (which are needed to supply functionality ASP can't provide) and aren't careful about how you design your application, the management nightmare grows. The bottom line is that no matter what approach you take, ASP code tends to become beastly, long, and incredibly difficult to debug—if you can even get ASP debugging working in your environment at all.

In ASP.NET, these problems don't exist. Web pages are written with traditional object-oriented concepts in mind. Your web pages contain controls that can be programmed against in a way similar to desktop applications. This means you don't need to combine a jumble of HTML markup and inline code. If you opt to use the code-behind approach when creating ASP.NET pages, the code and presentation are actually placed in two different files; simplifies code maintenance and allows you to separate the task of web-page design from the heavy-duty work of web coding.

Script Languages

At the time of its creation, ASP seemed like a perfect solution for desktop developers who were moving to the world of the Web. Rather than requiring programmers to learn a completely new language or methodology, ASP allowed developers to use familiar languages such as VBScript on a server-based programming platform. By leveraging the already-popular COM (Component Object Model) programming model as a backbone, these scripting languages also acted as a convenient vehicle for accessing server components and resources. But even though ASP was easy to understand for developers who were already skilled with scripting languages such as VBScript, this familiarity came with a price.

Performance wasn't the only problem. Every object or variable used in a classic ASP script is created as a *variant* data type. As most Visual Basic programmers know, variant data types are weakly typed. They require larger amounts of memory, are late-bound, and result in slower performance. Additionally, the compiler and development tools can't identify them at design time. This made it all but impossible to create a truly integrated IDE (integrated development environment) that could provide ASP programmers with anything like the powerful debugging, IntelliSense, and error checking found in Visual Basic and Visual C++. And without debugging tools, ASP programmers were hard-pressed to troubleshoot the problems in their scripts.

ASP.NET circumvents all these problems. For starters, ASP.NET web pages (and web services) are executed within the CLR (common language runtime), so they can be authored in any language that has a CLR-compliant compiler. No longer are you limited to using VBScript or JavaScript—instead, you can use modern object-oriented languages such as Visual Basic and C#.

It's also important to note that ASP.NET pages are not interpreted but are instead compiled into *assemblies* (the .NET term for any unit of compiled code). This is one of the most significant enhancements to Microsoft's web development model in ASP.NET 2.0. What actually happens behind the scenes is revolutionary. Even if you create your code in Notepad and copy it directly to a virtual directory on a web server, the application is dynamically compiled as soon as a client accesses it (in previous versions you had to precompile the application into a DLL), and it is cached for future requests. If any of the files are modified after this compilation process, the application is recompiled automatically the next time a client requests it.

The Death of COM

Though Microsoft claims undying support for COM, the technology that underlies the Windows operating system, and almost every application that runs on it, it's obvious that .NET is the start of a new path for modern development. Future versions of the Windows operating system (including the elusive Longhorn) will integrate the .NET Framework more deeply into the operating system kernel, making it the first-class language of all application development. And as COM applications wane in popularity and applications are converted to .NET, classic ASP will become a thing of the past. Even though .NET includes robust support for COM interoperability, the fact remains that classic ASP applications have no real place in a .NET world.

ASP.NET 1.0

Microsoft developers have described ASP.NET as their chance to "hit the reset button" and start from scratch with an entirely new, more modern development model. The traditional concepts involved in creating web applications still hold true in the .NET world. Each web application consists of web pages. You can render rich HTML and even use JavaScript, create components that encapsulate programming logic, and tweak and tune your applications using configuration settings. However, behind the scenes ASP.NET works quite differently than traditional scripting technologies such as classic ASP or PHP (PHP: Hypertext Preprocessor). It's also much more ambitious than JSP (Java Server Pages).

Some of the differences between ASP.NET and earlier web development platforms include the following:

- ASP.NET features a completely object-oriented programming model, which includes an event-driven, control-based architecture that encourages code encapsulation and code reuse.
- ASP.NET gives you the ability to code in any supported .NET language (including Visual Basic, C#, J#, and many other languages that have third-party compilers).
- ASP.NET is also a platform for building *web services*, which are reusable units of code that other applications can call across platform and computer boundaries. You can use a web service to do everything from web-enabling a desktop application to sharing data with a Java client running on Unix.
- ASP.NET is dedicated to high performance. ASP.NET pages and components are compiled on demand instead of being interpreted every time they're used. ASP.NET also includes, in ADO.NET, a fine-tuned data access model and flexible data caching to further boost performance.

These are only a few of the features, which include enhanced state management, practical data binding, dynamic graphics, and a robust security model. You'll look at these improvements in detail in this book and see what ASP.NET 2.0 adds to the picture.

Seven Important Facts About ASP.NET

If you're new to ASP.NET (or you just want to review a few fundamentals), you'll be interested in the following sections. They introduce seven touchstones of .NET development.

Fact 1: ASP.NET Is Integrated with the .NET Framework

The .NET Framework is divided into an almost painstaking collection of functional parts, with a staggering total of more than 7,000 *types* (the .NET term for classes, structures, interfaces, and other core programming ingredients). Before you can program any type of .NET application, you need a basic understanding of those parts—and an understanding of why things are organized the way they are.

The massive collection of functionality that the .NET Framework provides is organized in a way that traditional Windows programmers will see as a happy improvement. Each one of the thousands of data types in the .NET Framework is grouped into a logical, hierarchical container called a *namespace*. Different namespaces provide different features. Taken together, the .NET namespaces offer functionality for nearly every aspect of distributed development from message queuing to security. This massive toolkit is called the *class library*.

Interestingly, the way you use the .NET Framework classes in ASP.NET is the same as the way you use them in any other type of .NET application (including a stand-alone Windows application, a Windows service, a command-line utility, and so on). In other words, .NET gives the same tools to web developers that it gives to rich client developers.

If you've programmed extensively with ASP.NET 1.*x*, you'll find that the same set of classes is available in ASP.NET 2.0. The difference is that ASP.NET 2.0 adds even more classes to the mix, many in entirely new namespaces for features such as configuration, health monitoring, and personalization.

■ **Tip** One of the best resources for learning about new corners of the .NET Framework is the .NET Framework class library reference, which is part of the MSDN Help library reference. If you have Visual Studio 2005 installed, you can view the MSDN Help library by selecting Start ➤ Programs ➤ Microsoft Visual Studio 2005 ➤ Microsoft Visual Studio 2005 Documentation (the exact shortcut depends on your version of Visual Studio). Once you've loaded the help, you can find class reference information grouped by namespace under the .NET Development ➤ .NET Framework SDK ➤ Class Library Reference node.

Fact 2: ASP.NET Is Compiled, Not Interpreted

One of the major reasons for performance degradation in ASP scripts is that all ASP web-page code uses interpreted scripting languages. This means that when your application is executed, a scripting host on the server machine needs to interpret your code and translate it to lower-level machine code, line by line. This process is notoriously slow.

■ **Note** In fact, in this case the reputation is a little worse than the reality. Interpreted code is certainly slower than compiled code, but the performance hit isn't so significant that you can't build a professional website using ASP.

ASP.NET applications are always compiled—in fact, it's impossible to execute C# or VB .NET code without it being compiled first.

ASP.NET applications actually go through two stages of compilation. In the first stage, the C# code you write is compiled into an intermediate language called Microsoft Intermediate Language (MSIL) code, or just IL. This first step is the fundamental reason that .NET can be language-interdependent. Essentially, all .NET languages (including C#, Visual Basic, and many more) are compiled into virtually identical IL code. This first compilation step may happen automatically when the page is first requested, or you can perform it in advance (a process known as *precompiling*). The compiled file with IL code is an *assembly*.

The second level of compilation happens just before the page is actually executed. At this point, the IL code is compiled into low-level native machine code. This stage is known as *just-in-time* (JIT) compilation, and it takes place in the same way for all .NET applications (including Windows applications, for example). Figure 1-1 shows this two-step compilation process.

.NET compilation is decoupled into two steps in order to offer developers the most convenience and the best portability. Before a compiler can create low-level machine code, it needs to know what type of operating system and hardware platform the application will run on (for example, 32-bit or 64-bit Windows). By having two compile stages, you can create a compiled assembly with .NET code but still distribute this to more than one platform.

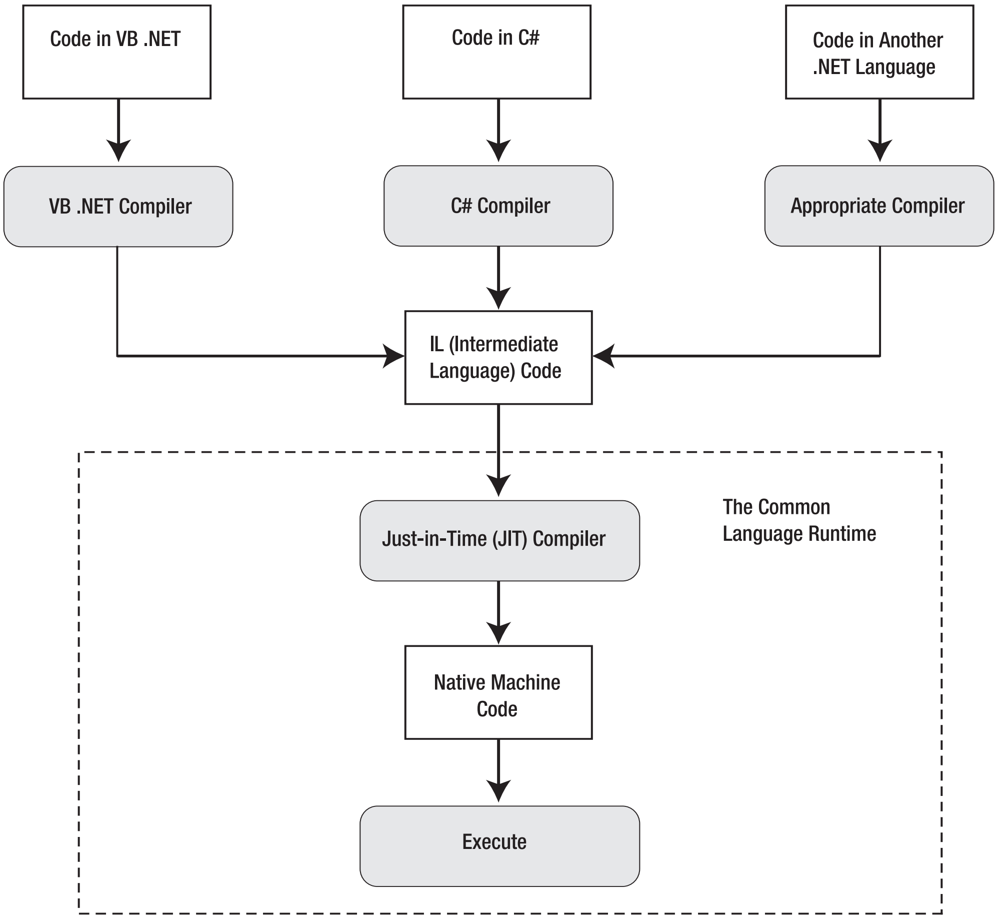

Code in VB .NET
Code in C#
Code in Another .NET Language
VB .NET Compiler
C# Compiler
Appropriate Compiler
IL (Intermediate Language) Code
Just-in-Time (JIT) Compiler
The Common Language Runtime
Native Machine Code
Execute

Figure 1-1. *Compilation in an ASP.NET web page*

■Note One day soon, this model may even help business programmers deploy applications to non-Microsoft operating systems such as Linux. This ambitious goal hasn't quite been realized yet, but if you'd like to try the first version of .NET for the Linux platform (complete with a work-in-progress implementation of ASP.NET), visit `http://www.go-mono.com` to download the latest version of this open-source effort.

Of course, JIT compilation probably wouldn't be that useful if it needed to be performed every time a user requested a web page from your site. Fortunately, ASP.NET applications don't need to be compiled every time a web page or web service is requested. Instead, the IL code is created once and regenerated only when the source is modified. Similarly, the native machine code files are cached in a system directory that has a path like c:\[WinDir]\Microsoft.NET\Framework\[Version]\Temporary ASP.NET Files, where [WinDir] in the Windows directory and [Version] is the version number for the currently installed version of the .NET Framework.

■Note Although benchmarks are often controversial, you can find an interesting comparison of Java and ASP.NET at `http://gotdotnet.com/team/compare`. Keep in mind that the real issues limiting performance are usually related to specific bottlenecks, such as disk access, CPU use, network bandwidth, and so on. In many benchmarks, ASP.NET outperforms other solutions because of its support for performance-enhancing platform features such as caching, not because of the speed boost that results from compiled code.

Although the compilation model in ASP.NET 2.0 remains essentially the same, it has one important change. The design tool (Visual Studio 2005) no longer compiles code by default. Instead, your web pages and services are compiled the first time you run them, which improves the debugging experience. To avoid the overhead of first-time compilation when you deploy a finished application (and prevent other people from tampering with your code), you can use a new *precompilation* feature, which is explained in Chapter 18.

Fact 3: ASP.NET Is Multilanguage

Though you'll probably opt to use one language over another when you develop an application, that choice won't determine what you can accomplish with your web applications. That's because no matter what language you use, the code is compiled into IL.

IL is a stepping-stone for every managed application. (A *managed application* is any application that's written for .NET and executes inside the managed environment of the CLR.) In a sense, IL is *the* language of .NET, and it's the only language that the CLR recognizes.

To understand IL, it helps to consider a simple example. Take a look at this example, written in VB .NET:

```
Namespace HelloWorld
    Public Class TestClass
        Private Shared Sub Main(Args() As String)
            Console.WriteLine("Hello World")
        End Sub
    End Class
End Namespace
```

This code shows the most basic application that's possible in .NET—a simple command-line utility that displays a single, predictable message on the console window.

Now look at it from a different perspective. Here's the IL code for the Main method:

```
.method public static void Main() cil managed
{
  .entrypoint
  .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() =
   ( 01 00 00 00 )
  // Code size       14 (0xe)
  .maxstack  8
  IL_0000:  nop
  IL_0001:  ldstr      "Hello World"
  IL_0006:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_000b:  nop
  IL_000c:  nop
  IL_000d:  ret
} // end of method TestClass::Main
```

It's easy enough to look at the IL for any compiled .NET application. You simply need to run the IL Disassembler, which is installed with Visual Studio and the .NET SDK (software development kit). Look for the file ildasm.exe in a directory like c:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin. Once you've loaded the program, use the File ➤ Open command, and select any DLL or EXE that was created with .NET.

If you're patient and a little logical, you can deconstruct the IL code fairly easily and figure out what's happening. The fact that IL is so easy to disassemble can raise privacy and code control issues, but these issues usually aren't of any concern to ASP.NET developers. That's because all ASP.NET code is stored and executed on the server. Because the client never receives the compiled code file, the client has no opportunity to decompile it. If it *is* a concern, consider using an obfuscator that scrambles code to try to make it more difficult to understand. (For example, an obfuscator might rename all variables to have generic, meaningless names such as f__a__234.) Visual Studio includes a scaled-down version of one popular obfuscator, called Dotfuscator.

The following code shows the same console application in C#:

```
namespace HelloWorld
{
    public class TestClass
    {
        private static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

If you compile this application and look at the IL code, you'll find that every line is semantically equivalent to the IL code generated from the VB .NET version. Although different compilers can sometimes introduce their own optimizations, as a general rule of thumb no .NET language outperforms any other .NET language, because they all share the same common infrastructure. This infrastructure is formalized in the CLS (Common Language Specification), which is described in the "The Common Language Specification" sidebar.

It's important to note that IL was recently adopted as an ANSI (American National Standards Institute) standard. This adoption could quite possibly spur the adoption of other common language frameworks. The Mono project at `http://www.go-mono.com` is an example of one such project.

THE COMMON LANGUAGE SPECIFICATION

The CLS defines the standard properties that all objects must contain in order to communicate with one another in a homogenous environment. To allow this communication, the CLR expects all objects to adhere to a specific set of rules.

The CLS is this set of rules. It defines many laws that all languages must follow, such as types, primitive types, method overloading, and so on. Any compiler that generates IL code to be executed in the CLR must adhere to all rules governed within the CLS. The CLS gives developers, vendors, and software manufacturers the opportunity to work within a common set of specifications for languages, compilers, and data types. As time goes on, you'll see more CLS-compliant languages and compilers emerge, although several are available so far.

Given these criteria, the creation of a language compiler that generates true CLR-compliant code can be complex. Nevertheless, compilers can exist for virtually any language, and chances are that there may eventually be one for just about every language you'd ever want to use. Imagine—mainframe programmers who loved COBOL in its heyday can now use their knowledge base to create web applications!

Fact 4: ASP.NET Runs Inside the Common Language Runtime

Perhaps the most important aspect of ASP.NET to remember is that it runs inside the runtime engine of the CLR. The whole of the .NET Framework—that is, all namespaces, applications, and classes—are referred to as *managed* code. Though a full-blown investigation of the CLR is beyond the scope of this chapter, some of the benefits are as follows:

Automatic memory management and garbage collection: Every time your application creates an instance of a class, the CLR allocates space on the *managed heap* for that object. However, you never need to clear this memory manually. As soon as your reference to an object goes out of scope (or your application ends), the object becomes available for garbage collection. The garbage collector runs periodically inside the CLR, automatically reclaiming unused memory for inaccessible objects. This model saves you from the low-level complexities of C++ memory handling and from the quirkiness of COM reference counting.

Type safety: When you compile an application, .NET adds information to your assembly that indicates details such as the available classes, their members, their data types, and so on. As a result, your compiled code assemblies are completely self-sufficient. Other people can use them without requiring any other support files, and the compiler can verify that every call is valid at runtime. This extra layer of safety completely obliterates low-level errors such as the infamous buffer overflow in C++.

Extensible metadata: The information about classes and members is only one of the types of metadata that .NET stores in a compiled assembly. *Metadata* describes your code and allows you to provide additional information to the runtime or other services. For example, this metadata might tell a debugger how to trace your code, or it might tell Visual Studio how to display a custom control at design time. You could also use metadata to enable other runtime services (such as web methods or COM+ services).

Structured error handling: If you've ever written any moderately useful Visual Basic or VBScript code, you'll most likely be familiar with the limited resources these languages offer for error handling. With structured exception handling, you can organize your error-handling code logically and concisely. You can create separate blocks to deal with different types of errors. You can also nest exception handlers multiple layers deep.

Multithreading: The CLR provides a pool of threads that various classes can use. For example, you can call methods, read files, or communicate with web services asynchronously, without needing to explicitly create new threads.

Figure 1-2 shows a high-level look at the CLR and the .NET Framework.

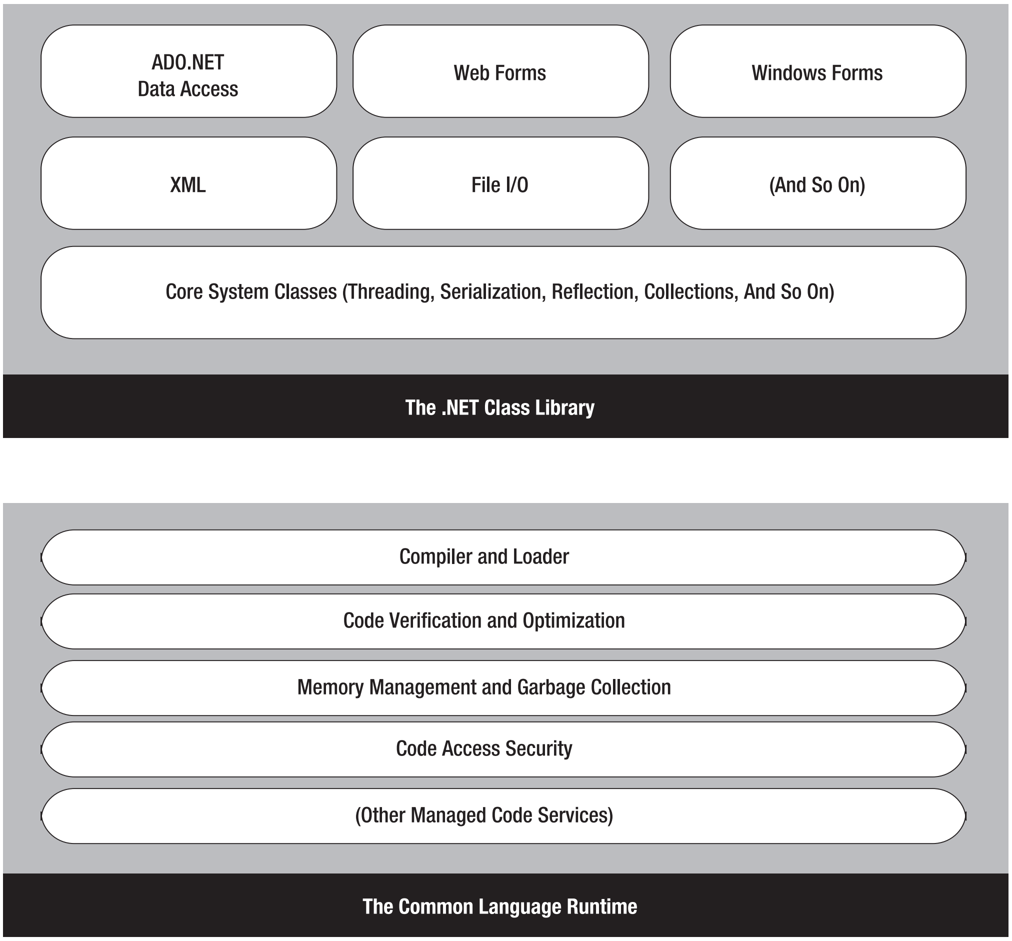

ADO.NET
Data Access
Web Forms
Windows Forms
XML
File I/O
(And So On)
Core System Classes (Threading, Serialization, Reflection, Collections, And So On)
The .NET Class Library
Compiler and Loader
Code Verification and Optimization
Memory Management and Garbage Collection
Code Access Security
(Other Managed Code Services)
The Common Language Runtime

Figure 1-2. *The CLR and .NET Framework*

Fact 5: ASP.NET Is Object-Oriented

ASP provides a relatively lightweight object model, albeit one that is extensible using heavy COM objects. It provides a small set of objects; these objects are really just a thin layer over the raw details of HTTP and HTML. On the other hand, ASP.NET is truly object-oriented. Not only does your code have full access to all objects in the .NET Framework, but you can also exploit all the conventions of an OOP (object-oriented programming) environment, such as encapsulation and inheritance. For example, you can create reusable classes, standardize code with interfaces, and bundle useful functionality in a distributable, compiled component.

One of the best examples of object-oriented thinking in ASP.NET is found in *server-based controls*. Server-based controls are the epitome of encapsulation. Developers can manipulate server controls programmatically using code to customize their appearance, provide data to display, and even react to events. The low-level HTML details are hidden away behind the scenes. Instead of forcing the developer to write raw HTML manually, the control objects render themselves to HTML when the page is finished rendering. In this way, ASP.NET offers server controls as a way to abstract the low-level details of HTML and HTTP programming.

Here's a quick example with a standard HTML text box in an ASP.NET web page:

```
<input type="text" id="myText" runat="server" />
```

With the addition of the runat="server" attribute, this static piece of HTML becomes a fully functional server-side control that you can manipulate in your code. You can now work with server-side events that it generates, set attributes, and bind it to a data source.

For example, you can set the text of this box when the page first loads using the following code:

```
Private Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        myText.Value = "Hello World!"
End Sub
```

Technically, this code sets the Value property of an HtmlInputText object. The end result is that a string of text appears in a text box on the HTML page that's rendered and sent to the client.

HTML CONTROLS VS. WEB CONTROLS

When ASP.NET was first created, two schools of thought existed. Some ASP.NET developers were most interested in server-side controls that matched the existing set of HTML controls exactly. This approach allows you to create ASP.NET web-page interfaces in dedicated HTML editors, and it provides a quick migration path for existing ASP pages. However, another set of ASP.NET developers saw the promise of something more—rich server-side controls that didn't just emulate individual HTML tags. These controls might render their interface from dozens of distinct HTML elements while still providing a simple object-based interface to the programmer. Using this model, developers could work with programmable menus, calendars, data lists, and validators.

After some deliberation, Microsoft decided to provide both models. You've already seen an example of HTML server controls, which map directly to the basic set of HTML tags. Along with these are ASP.NET *web controls*, which provide a higher level of abstraction and more functionality. In most cases, you'll use HTML server-side controls for backward compatibility and quick migration and use web controls for new projects.

ASP.NET web control tags always start with the prefix *asp:* followed by the class name. For example, the following snippet creates a text box and a check box:

```
<asp:TextBox ID="myASPText" Text=”Hello ASP.NET TextBox" runat="server" />
<asp:CheckBox ID="myASPCheck" Text="My CheckBox" runat="server" />
```

Again, you can interact with these controls in your code, as follows:

```
myASPText.Text = "New text"
myASPCheck.Text = "Check me!"
```

Notice that the Value property you saw with the HTML control has been replaced with a Text property. The HtmlInputText.Value property was named to match the underlying value attribute in the HTML <input> tag. However, web controls don't place the same emphasis on correlating with HTML syntax, so the more descriptive property name Text is used instead.

The ASP.NET family of web controls includes complex rendered controls (such as the Calendar and TreeView), along with more streamlined controls (such as TextBox, Label, and Button), which map closely to existing HTML tags. In the latter case, the HTML server-side control and the ASP.NET web control variants provide similar functionality, although the web controls tend to expose a more standardized, streamlined interface. This makes the web controls easy to learn, and it also means they're a natural fit for Windows developers moving to the world of the Web, because many of the property names are similar to the corresponding Windows controls.

Fact 6: ASP.NET Is Multidevice and Multibrowser

One of the greatest challenges web developers face is the wide variety of browsers they need to support. Different browser brands, versions, and configurations differ in their support of HTML. Web developers need to choose whether they should render their content according to HTML 3.2, HTML 4.0, or something else entirely—such as XHTML 1.0 or even WML (Wireless Markup Language) for mobile devices. This problem, fueled by the various browser companies, has plagued developers since the World Wide Web Consortium proposed the first version of HTML. Life gets even more complicated if you want to use a client-side HTML extension such as JavaScript to create a more dynamic page or provide validation.

ASP.NET addresses this problem in a remarkably intelligent way. Although you can retrieve information about the client browser and its capabilities in an ASP.NET page, ASP.NET actually encourages developers to ignore these considerations and use a rich suite of web server controls. These server controls render their HTML adaptively by taking the client's capabilities into account. One example is ASP.NET's validation controls, which use JavaScript and DHTML (Dynamic HTML) to enhance their behavior if the client supports it. This allows the validation controls to show dynamic error messages without the user needing to send the page back to the server for more processing. These features are optional, but they demonstrate how intelligent controls can make the most of cutting-edge browsers without shutting out other clients. Best of all, you don't need any extra coding work to support both types of client.

■ **Note** Unfortunately, ASP.NET 2.0 still hasn't managed to integrate mobile controls into the picture. As a result, if you want to create web pages for *smart devices* such as mobile phones, PDAs (personal digital assistants), and so on, you need to use a similar but separate toolkit. The architects of ASP.NET originally planned to unify these two models so that the standard set of server controls could render markup using a scaled-down standard such as WML or HDML (Handheld Device Markup Language) instead of HTML. However, this feature was cut late in the beta cycle.

Fact 7: ASP.NET Is Easy to Deploy and Configure

One of the biggest headaches a web developer faces during a development cycle is deploying a completed application to a production server. Not only do the web-page files, databases, and components need to be transferred, but you also need to register components and re-create a slew of configuration settings. ASP.NET simplifies this process considerably.

Every installation of the .NET Framework provides the same core classes. As a result, deploying an ASP.NET application is relatively simple. In most cases, you simply need to copy all the files to a virtual directory on a production server (using an FTP program or even a command-line command like XCOPY). As long as the host machine has the .NET Framework, there are no time-consuming registration steps.

Distributing the components your application uses is just as easy. All you need to do is copy the component assemblies when you deploy your web application. Because all the information about your component is stored directly in the assembly file metadata, there's no need to launch a registration program or modify the Windows registry. As long as you place these components in the correct place (the Bin subdirectory of the web application directory), the ASP.NET engine automatically detects them and makes them available to your web-page code. Try that with a traditional COM component!

Configuration is another challenge with application deployment, particularly if you need to transfer security information such as user accounts and user privileges. ASP.NET makes this deployment process easier by minimizing the dependence on settings in IIS (Internet Information Services). Instead, most ASP.NET settings are stored in a dedicated web.config file. The web.config file is placed in the same directory as your web pages. It contains a hierarchical grouping of application settings stored in an easily readable XML format that you can edit using nothing more than a text editor such as Notepad. When you modify an application setting, ASP.NET notices that change and smoothly restarts the application in a new application domain (keeping the existing application domain alive long enough to finish processing any outstanding requests). The web.config file is never locked, so it can be updated at any time.

ASP.NET 2.0: The Story Continues

When Microsoft released ASP.NET 1.0, even it didn't anticipate how enthusiastically the technology would be adopted. ASP.NET quickly became the standard for developing web applications with Microsoft technologies and a heavy-hitting competitor against all other web development platforms.

■**Note** Adoption statistics are always contentious, but the highly regarded Internet analysis company Netcraft (`http://www.netcraft.com`) suggests that ASP.NET usage doubled in one year and that it now runs on more web servers than JSP. This survey doesn't weigh the relative size of these websites, but ASP.NET powers the websites for a significant number of Fortune 1000 companies.

It's a testament to the good design of ASP.NET 1.0 and 1.1 that few changes in ASP.NET 2.0 are fixes for existing features. Instead, ASP.NET 2.0 keeps the same underlying plumbing and concentrates on adding new, higher-level features. In other words, ASP.NET 2.0 contains more features, frills, and tools, all of which increase developer productivity. The goal, as stated by the ASP.NET team, is to reduce the number of lines of code you need to write by 70 percent.

■**Note** In reality, professional web applications probably won't achieve the 70 percent code reduction. However, you'll probably be surprised to find new features that you can drop into your applications with only a few minor tweaks. And unlike many half-baked frills, you won't need to abandon these features and start from scratch to create a real-world application. Instead, you can plug your own modules directly into the existing framework, saving time and improving the flexibility and reusability of the end result.

Officially, ASP.NET 2.0 is backward compatible with ASP.NET 1.0. In reality, 100 percent backward compatibility never exists, because correcting bugs and inconsistencies in the language can change how existing code works. Microsoft maintains a list of the breaking changes (most of which are very obscure) at `http://www.gotdotnet.com/team/changeinfo/Backwards1.1to2.0`. However, you're unlikely to run into a problem when migrating an ASP.NET 1.*x* project to ASP.NET 2.0. It's much more likely that you'll find some cases where the old way of solving a problem still works but ASP.NET 2.0 introduces a much better approach. In these cases, it's up to you whether to defer the change or try to reimplement your web application to take advantage of the new features.

Of course, ASP.NET 2.0 isn't just about adding features. It also streamlines performance and simplifies configuration with a new tool called the WAT (website administration tool). The following sections introduce some of the most important changes in the different parts of the .NET Framework.

Visual Basic 2005

Visual Basic 2005 has several new language features. Some of these are exotic features that only a language aficionado will love, and others are more generally useful. The new features include the following:

Partial classes: Partial classes allow you to split a class into two or more source code files. This feature is primarily useful for hiding messy details you don't need to see. Visual Studio uses partial classes in some project types to tuck automatically generated code out of sight.

Generics: Generics allow you to create classes that are flexible enough to work with different class types but still support strong type checking. For example, you could code a collection class using generics that can store any type of object. When you create an instance of the collection, you "lock it in" to the class of your choice so that it can store only a single type of data. The important part in this example is that the locking happens when you *instantiate* the collection class, not when you code it.

Anonymous methods: Anonymous methods allow you to define a block of code on the fly, inside another method. You can use this technique to quickly hook up an event handler.

The My object: This object encapsulates some of the most common functionality used by developers. It exposes several different objects such as My.Application and My.Computer.

You'll see partial classes in action in Chapter 2, and you'll use generic classes with collections later in this book.

Visual Studio 2005

Microsoft provided two separate design tools for creating web applications with ASP.NET 1.*x*—the full-featured Visual Studio .NET and the free Web Matrix. Professional developers strongly favored Visual Studio .NET, but Web Matrix offered a few innovative features of its own. Because Web Matrix included its own scaled-down web server, programmers could create and test web applications without needing to worry about configuring virtual directories on their computer using IIS.

With .NET 2.0, Web Matrix disappears, but Visual Studio steals some of its best features, including the integrated web server, which lets you get up and running with a test website in no time, without the need for IIS or virtual directories on your development machine.

Another welcome change in Visual Studio 2005 is the support for different coding models. While Visual Studio .NET 2003 locked developers into one approach, Visual Studio 2005 supports a range of different coding models, making it a flexible, all-purpose design tool. That means you can choose to put your HTML tags and event-handling code in the same file, or in separate files, without compromising your ability to use Visual Studio and benefit from helpful features such as IntelliSense. (You'll learn about this distinction in Chapter 2.) You can also use more than one programming language in the same project, mixing C# web pages with VB web pages, or vice versa.

ASP.NET 2.0

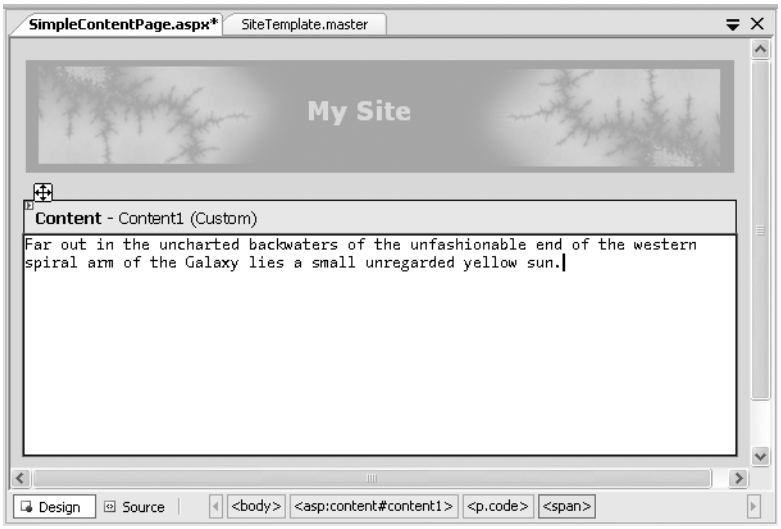
For the most part, this book won't distinguish between the features that are new in ASP.NET 2.0 and those that have existed since ASP.NET 1.0. However, in the next few sections you'll tour some of the highlights.

Master Pages

Need to implement a consistent look across multiple pages? With *master pages*, you can define a template and reuse it effortlessly. For example, you could use a template to ensure that every web page in your application has the same header, footer, and navigation controls.

Master pages define specific editable regions, called *content regions*. Each page that uses the master page acquires its layout and its fixed elements automatically and supplies the content for just these regions.

Figure 1-3 shows an example content page at design time. The master page supplies the header and formatting of the outlying page. The content page is limited to inserting additional HTML and web controls in a specific region.


SimpleContentPage.aspx*
SiteTemplate.master
My Site
Content - Content1 (Custom)
Design
Source
<body> <asp:content#content1> <p.code> <span>

Far out in the uncharted backwaters of the unfashionable end of the western spiral arm of the Galaxy lies a small unregarded yellow sun.

Figure 1-3. *A content page at design time*

On a related note, ASP.NET also adds a new theme feature, which lets you define a standardized set of appearance characteristics for web controls. Once you've defined these formatting presets, you can apply them across your website for a consistent look.

Interestingly, you can set both master and themes pages at runtime. This means you can write code to apply different themes and master pages depending on the type of user or on the user's preferences. In this way, you can use master pages and themes not just to standardize your website but to make it customizable. You'll learn about master pages and themes in Chapter 15.

Data Source Controls

Tired of managing the retrieval, format, and display of your data? With the new data source control model, you can define how your page interacts with a data source *declaratively* in your page, rather than writing the same boilerplate code to access your data objects. Best of all, this feature doesn't force you to abandon good component-based design—you can bind to a custom data component just as easily as you bind directly to the database.

Here's how the new data-binding model works at its simplest. First, drop the GridView onto a page using Visual Studio, or code it by hand using this tag:

```
<asp:GridView id="MyDataGrid" runat="server"/>
```

Next, you need to add the data source, which will fetch the rows you're interested in and make them available to the GridView. This simple example uses the SqlDataSource to connect directly to a SQL Server database, but a professional application will usually use the ObjectDataSource to go through a separate layer of custom components. To create the SqlDataSource tag, you need a few details, including the query used to retrieve the records and the connection string used to access the database. You can walk through this process with a Visual Studio wizard, or you can code it by hand. Either way, you'll end up with something like this (assuming that the SQL Server database you want to connect to is on the current computer and supports Windows authentication):

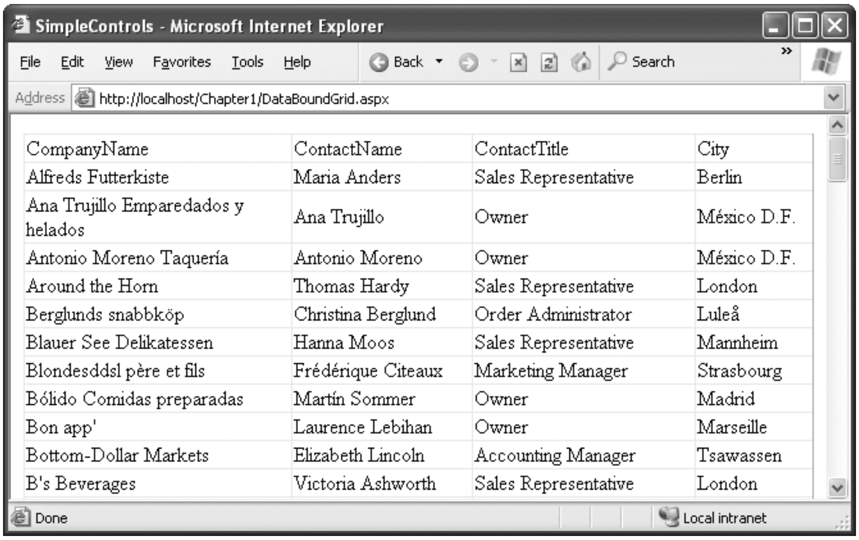
```
<asp:SqlDataSource ID="CustomersList" Runat="server"
  SelectCommand="SELECT CompanyName, ContactName, ContactTitle, City FROM Customers"
  ConnectionString=
  "Data Source=127.0.0.1;Integrated Security=SSPI;Initial Catalog=Northwind">
</asp:SqlDataSource>
```

This data source defines a connection to the Northwind database and a Select operation that retrieves all the records in the Customers table.

Finally, you need to bind the data source to the GridView. To do this, set the GridView.DataSourceID property to the name of the SqlDataSource (in this example, CustomersList). You can do this in code or using the Visual Studio properties window, in which case you modify the GridView tag to look like this:

```
<asp:GridView id="MyDataGrid" DataSourceID="CustomersList" runat="server"/>
```

Without writing any code or adding special formatting to the GridView control (and there are a lot of options for doing exactly that), you'll see the bare-bones table in Figure 1-4. On top of this basic representation, you can define values for features such as font styling, background colors, header styles, and much more. You can also enable features for column-based sorting, paging (splitting a table over multiple pages), selecting, and editing.


SimpleControls - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Address http://localhost/Chapter1/DataBoundGrid.aspx
Done
Local intranet

CompanyName	ContactName	ContactTitle	City
Alfreds Futterkiste	Maria Anders	Sales Representative	Berlin
Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	México D.F.
Antonio Moreno Taquería	Antonio Moreno	Owner	México D.F.
Around the Horn	Thomas Hardy	Sales Representative	London
Berglunds snabbköp	Christina Berglund	Order Administrator	Luleå
Blauer See Delikatessen	Hanna Moos	Sales Representative	Mannheim
Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	Strasbourg
Bólido Comidas preparadas	Martín Sommer	Owner	Madrid
Bon app'	Laurence Lebihan	Owner	Marseille
Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	Tsawassen
B's Beverages	Victoria Ashworth	Sales Representative	London

Figure 1-4. *A simple data-bound grid*

Along with the GridView, ASP.NET 2.0 also adds other new controls for displaying data, including the DetailsView and FormView controls. Both controls can act as a record browser, showing detailed information for a single record at a time. They also support editing. You'll learn about the new data features throughout Part 2.

Personalization

Most web applications deal extensively with user-specific data. For example, if you're building an e-commerce site, you might need to store and retrieve the current user's address, viewing preferences, shopping basket, and so on. ASP.NET 1.*x* allowed you to cache this information for a short amount of time, but it was still up to you to write this information to a database if you needed it for a longer period of time and then retrieve it later.

ASP.NET 2.0 addresses this limitation with *personalization*, an API for dealing with user-specific information that's stored in a database. The idea is that ASP.NET creates a profile object where you can access the user-specific information at any time. Behind the scenes, ASP.NET takes care of the tedious work of retrieving the profile data when it's needed and saving the profile data when it changes.

Most serious developers will quickly realize that the default implementation of personalization is a one-size-fits-all solution that probably won't suit their needs. For example, what if you need to use existing database tables, store encrypted information, or customize how large amounts of data are cached to improve performance? Interestingly, you can customize personalization to suit your needs by building your own personalization provider. This allows you to use the convenient personalization features but still control the low-level details. Of course, the drawback is that you're still responsible for some of the heavy lifting (no more 70 percent code reduction), but you gain the flexibility and consistency of the profile model. You'll learn about personalization in Chapter 24.

■ **Tip** Many of the features in ASP.NET 2.0 work through an abstraction called the *provider model*. The beauty of the provider model is that you can use the simple providers to build your page code. If your requirements change, you don't need to change a single page—instead, you simply need to create a custom provider. The provider model is useful enough that a similar organization pattern was used for similar handcrafted solutions in the first edition of this book, before ASP.NET 2.0 appeared.

Security and Membership

One of the most useful features in ASP.NET 1.*x* was forms authentication, a cookie-based system for tracking authenticated users. Although forms authentication worked perfectly well for securing a website, it was still up to each web developer to write the code for authenticating the user in a login page. And forms authentication didn't provide any functionality for user authorization (testing if the current user has a certain set of permissions), which meant developers were forced to add these features from scratch if they were needed.

ASP.NET 2.0 addresses both of these shortcomings by extending forms authentication with new features. First, ASP.NET includes automatic support for tracking user credentials, securely storing passwords, and authenticating users in a login page. You can customize this functionality based on your existing tables, or you can simply point ASP.NET to your database server and let it manage everything. Additionally, ASP.NET includes a handful of new controls for managing security, allowing users to log in, register, and retrieve passwords. You can let these controls work on their own without any custom code, or you can configure them to match your requirements.

Finally, ASP.NET adds support for authorization with a *membership* API. Membership allows you to use role-based authorization. You map your users into different groups (like Guest, Administrator, SalesEmployee) and then you test if a user is a member of the right group before allowing a specific action. Best of all, membership plugs right into the forms-based security infrastructure. You'll learn much more in Part 4.

Rich Controls

All in all, ASP.NET introduces more than 40 controls. Many of these controls support new features, such as the dedicated security controls and web parts controls for portals. You'll also find a handy wizard and MultiView control that allow you to create pages with multiple views. But the two most impressive controls are probably the new TreeView and JavaScript-powered Menu.

The TreeView allows you to show a hierarchical, collapsible tree view of data with extensive customization. Figure 1-5 shows a few of your menu options for outfitting the TreeView with different node pictures.

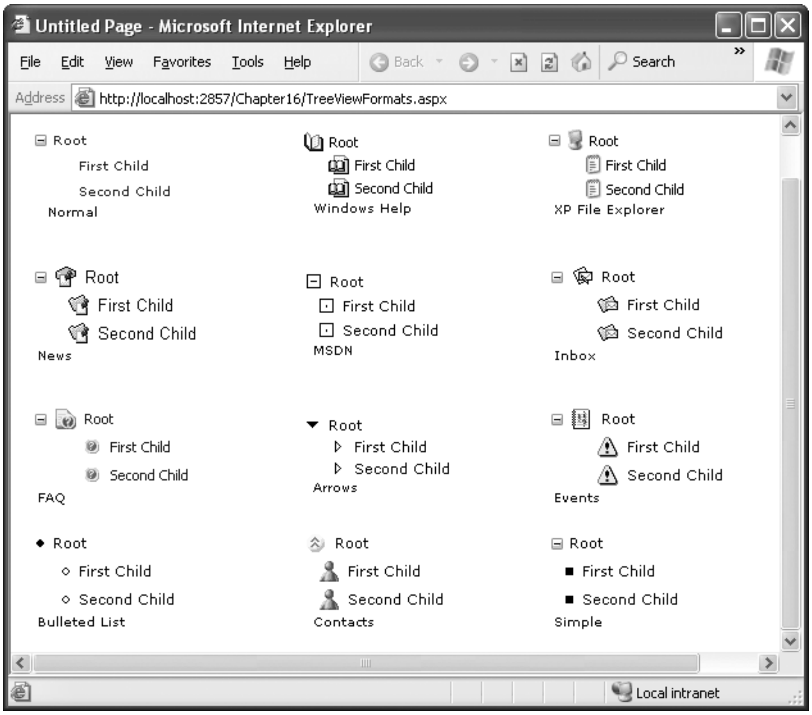


Figure 1-5. *Node styles with the new TreeView control*

The new Menu control also deals with displaying hierarchical data, but it renders itself as a JavaScript-powered fly-out menu. As you move the mouse, the appropriate submenu appears, superimposed over the current page (see Figure 1-6).

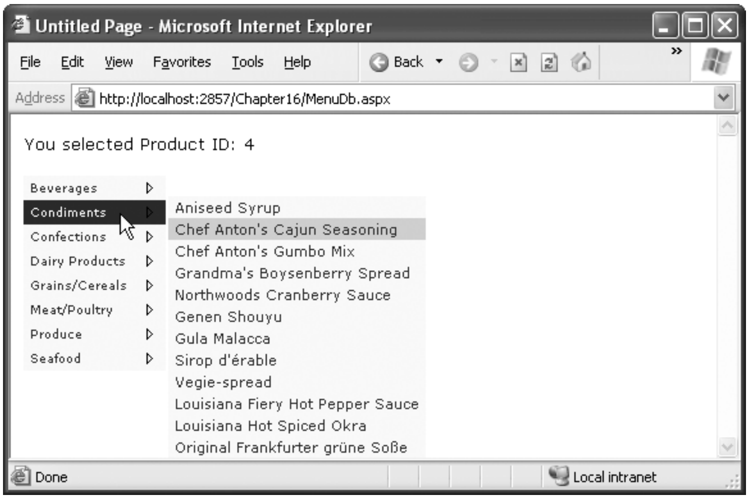

Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://localhost:2857/Chapter16/MenuDb.aspx
You selected Product ID: 4
Beverages
Condiments
Confections
Dairy Products
Grains/Cereals
Meat/Poultry
Produce
Seafood
Aniseed Syrup
Chef Anton's Cajun Seasoning
Chef Anton's Gumbo Mix
Grandma's Boysenberry Spread
Northwoods Cranberry Sauce
Genen Shouyu
Gula Malacca
Sirop d'érable
Vegie-spread
Louisiana Fiery Hot Pepper Sauce
Louisiana Hot Spiced Okra
Original Frankfurter grüne Soße
Done
Local intranet

Figure 1-6. *The dynamic Menu control*

Both the TreeView and the Menu are useful for displaying arbitrary data and for showing a navigation tree so that users can surf from one page to another on your website. To make navigation even easier, ASP.NET also adds an optional model for creating site maps that describe your website. Once you create a site map, you can use it with the new navigation seamlessly. Best of all, from that point on you can change the structure of your website or add new pages without needing to modify anything other than a single site-map file. You'll see the navigation controls in action in Chapter 16.

Web Parts

One common type of web application is the *portal*, which centralizes different information using separate panes on a single web page. Although you could create a portal website in ASP.NET 1.*x*, you needed to do it by hand. In ASP.NET 2.0, a new web parts feature makes life dramatically easier with a prebuilt portal framework. And what a model it is—complete with a flow-based layout, configurable views, and even drag-and-drop support. Indeed, if you're planning to create a web portal with these features, it's safe to say that ASP.NET 2.0 will deliver the promised 70 percent code savings. You'll see more of this advanced feature in Chapter 31.

Administration

To configure an application in ASP.NET 1.*x*, you needed to edit a configuration file by hand. Although this process wasn't too difficult, ASP.NET 2.0 streamlines it with a dedicated web administration tool that works through a web-page interface. This tool, called the WAT, is particularly useful if you're also using the personalization and membership features. That's because the WAT gives you a convenient (if slightly sluggish) interface for defining user-specific data, adding users, assigning users to roles, and more. You'll take your first look at the WAT in Chapter 5.

Summary

So far, you've only just scratched the surface of the features and frills that are provided in ASP.NET and the .NET Framework. You've taken a quick look at the high-level concepts you need to understand in order to be a competent ASP.NET programmer. You've also previewed the new features that ASP.NET 2.0 offers. As you continue through this book, you'll learn much more about the innovations and revolutions of ASP.NET 2.0 and the .NET Framework.